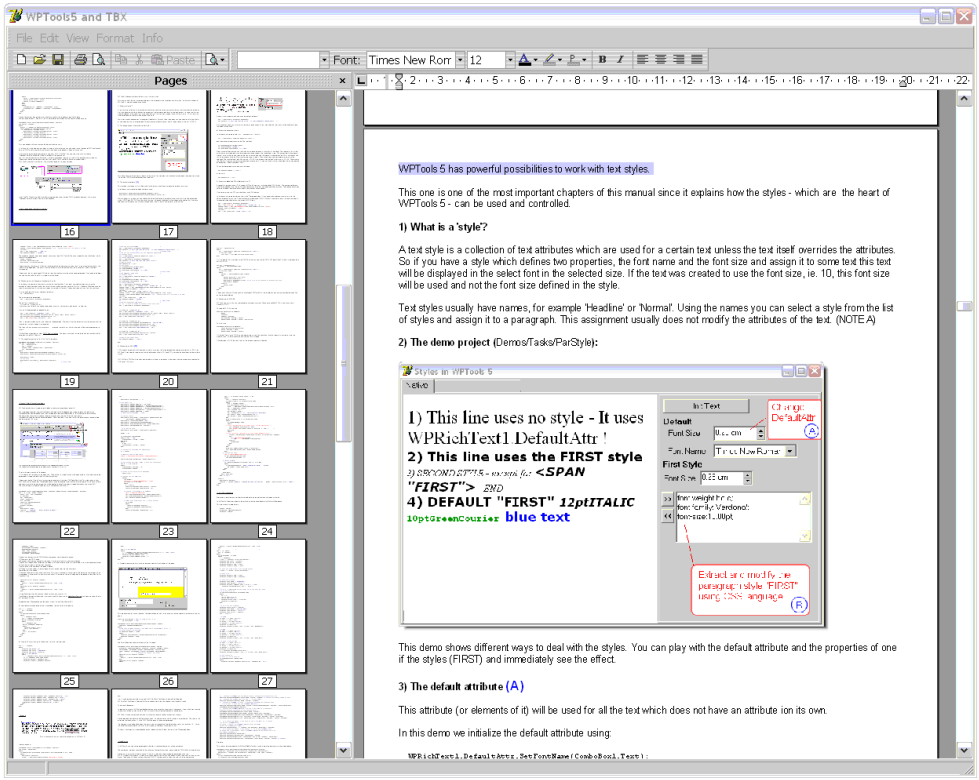
WPTools 5

GUIDE



05.05.2005

Table of Contents

1 What is WPTools?

The History

The first version of our product was released on the Delphi market in January 1996. Over the years it has evolved further to become what you now see. First of all, HTML and WYSIWYG support was added, later we also added the page layout view and fast zooming. At the beginning of 2004 version 4.22 was released - it was the last release of a WPTools version which was still partially based on the RTF Engine created in 1996.

During 2003 and 2004 the WPCubed GmbH, managed by Julian Ziersch, developed a new word processing engine. This new engine was constructed to provide solutions for the wide variety of demands which were raised over the last 8 years and addresses issues which could not be solved within the framework of the old WPTools engine.

WPTools has been quite successful over the last 8 years and was used in a multitude of projects, both large and small. It is the only text editor which was developed in native Delphi and which supports WYSIWYG page layout view (with WYSIWYG header and footer), including support for tab stops. The competition continues to struggle to achieve the standard set by WPTools, thus demonstrating that the original concept was very good. So why was this rewrite necessary?

In-depth modifications were required to add support for new features, such as nested tables. Plus, .NET compliancy made it necessary to remove pointer aritmetic completely. This had been very important in 1996 to enhance performace and because the compiler did not support arrays with variable lengths. Furthermore, some parts of the programming interface had become redundant over the years and last, but not least, CSS and XML development brought new ideas to word processing, which could only be implemented in a complete re-write.

WPTools 4 had heaps of functionality, but a great deal of it was developed in an ad-hoc manner - a problem arose and we provided a solution. But the various solutions provided did not necessarily gel consistently into an integrated framework and the planned architecture of the engine.

In the end we decided it would be a good idea to take advantage of the opportunity to develop a new RTF-Engine to fulfill the needs which had arisen within a structured environment.

What does WPTools 5 do?

a) Word processing (WYSIWYG, page layout view, headers + footers, tab stops, paragraph styles, UNICODE).
b) Special text editing tasks - most importantly, editing one text stream with multiple editors which can be on different forms.
c) Reporting - WPTools has the ability to dynamically create and display text during page formatting. This makes it possible to display calculated sums in dynamic table footers. Because the display changes after each reformat, this is an ideal solution for creating invoices since the report is completely editable!
d) Preview - The developer can insert pages from any source (owner drawn) into the displayed pages. This way it is possible to mix these pages with RTF pages and preview and print them all at once. It is also possible to have an editable text, followed by a report created by the specialized reporting engine and preview everything in the same editor!
e) Spread sheet - WPTools has incredibly powerful and fast table support. It is fast enough to hold thousands of rows!
f) Mail merge - It is possible to merge in text, plain or formatted and also images.
g) HTML editing - WPTools supports additive paragraph attributes which makes it perfect for this task. In contrast to other solutions which completely import or export the HTML properties, the form of the imported HTML text remains intact.
h) Automatic text creation. The new TableAdd function is very easy to use but still very powerful. As we prove in our "ThreadSave" demo the text creation can take place in sub threads.

2 WPTools Version 5

WPTools Version 5 has an entirely new RTF Engine. This makes it much easier to work with the paragraph text and offers an object-oriented approach whenever such an approach is benefitial.

Please read chapter "Release notes" and "Data Structures".

The new data model supports a separation between data and editing logic. This makes it possible to attach multiple editors to one text object in order to view different parts of the same text. This feature can be used to create a split view of the same text or, even more interesting, to link different editors to specific pages. This feature can also be used to edit text paths in DTP applications.

The result is a suite of components with a combination of features you will not find available in any other single component package.

New concepts **found in WPTools Version 5 include:**

- storage of character attributes in a special record which is assigned to the character which uses these particular attributes. This makes it easy to copy characters and assign new attributes. Other products use start and end tags to store character attributes with the danger of inconsistency if a tag 'gets lost'. Since WPTools caches the attributes, it does not consume a lot memory for them!
- paragraph storage in a linked list with sub-elements (children) - this reflects the document structure of HTML and makes it possible to support HTML/CSS editing.
- inherited attributes - in WPTools paragraph and character attributes can be added and removed. If an attribute is not defined the 'default' or 'inherited' value is used instead.

For *application designers* WPTools Version 5 supports

- different types of multiple page layout which can be displayed quickly
- fast zooming
- the possibility of editing ONE text in TWO (or more) windows (splitscreen)
- effective property dialogs (NEW: The dialogs shade 'undefined' values)
- integrated localization
- and all important action classes are included.

For *word processing* WPTools Version 5 includes

- a vast number of paragraph and character attributes for text (numbering, alignment, indents, spacing - the usual RTF features, but also color and style for various underlines, different border styles, background colors)
- strong table support - including the ability to nest tables (table within a table) and merging of table columns and table rows, with support for page breaks within table rows
- Repeated table header and footer rows
- pictures with text flow on one or (new!) both sides
- powerful support for headers and footers
- extensive support for paragraph styles
- improved support for tab stops - including support for leading signs (.....1.00).
- insertion of foreign pages - i.e. pages which are painted by an owner-provided function. These pages appear in the text, but cannot be accessed with the cursor
- WYSIWYG
- and different layout modes - so powerful that the preview dialog now uses a standard editor which is switched to read only. It still allows thumbnails and side-by-side page layout views!

For *HTML/CSS editing* WPTools Version 5 enables

- loading of HTML/CSS files
- saving of HTML/CSS files

- and powerful CSS support - the word processing engine supports inherited attributes and nested tables and paragraphs.

For *developers* WPTools Version 5 is an improvement because it

- is not bloatware - in fact WPTools V5 is smaller than WPTools 4!
- offers effective memory management. It also supports many more paragraph and character attributes without consuming more memory than the previous version (under the same OS).
- is safe for multi-threaded applications
- allows the addition of new paragraph attributes
- makes it possibile to insert 'external' or 'owner drawn' pages into a document
- offers support of most RTF features in HTML/CSS file format and vice versa. (Notable exception: Tab stops cannot be coded into HTML/CSS.)
- makes it possible to share styles and other properties (color palette, font table) between texts. This makes it easier to work with two synchronized RTF texts.
- allows linking of editors to text paths (the text is never copied between these editors).
- provides support for Delphi 5, 6, 7 and 2005
- provides support for C++Builder 5 and 6
- provides an effective storage format which produces the minimum amount of RTF and HTML/CSS code possible.

Optional in WPTools PREMIUM:

- support for columns (newspaper layout)
- editing and printing of footnotes and end notes
- and positioned paragraphs (text frames).

Other new products planned for in WPTools family:

- WPTools5.NET with support for C# and other .NET languages (winforms)
- WPTools5.DLL an embedded word processor with support for all languages which support DLLs
- WPSpell - spell checking highly integrated into the WPTools enviroment

We hope that you are as satisfied with WPTools Version 5 as we are and that the improvement in your product is worth the effort which may be required for the upgrade to this new version.
WPTools 5 was built to be the most powerful word processor available on the component market. In addition, it also works as a HTML editor, offering convenient word processing features, while still working as closely to the HTML document model as possible.

3 Release notes

For questions about WPTools 5 please use our web based forum:
Discussion
FAQ: Questions when upgrading
FAQ: General

26.4.2005 - V5.0 RELEASE 15

+ WPRichText.ParStyles.SaveToFile / LoadFromFile to make it easy to load/save style sheet in WPCSS format
+ WPRichText.ParStyle.SetWPCSS(s) applieas a string which was created by GetWPCSS
* protected text is now also locked for attribute changes (unless ppDontProtectAttributes is used in ProtectedProp)
* updated demo ParStyles
* updated RTF font/charset writing. The resulting file will be a lot smaller
+ New procedure DeletePage - deletes the text on a certain page.
This is a very complicated function which also handles multipage table rows!
- fixed broken 'space before'

- fixed a possible memory leak when using procedure TextObjects.Insert() with a TGraphic object.
  (the passed object will be automatically freed)
- fixed memory leak in TextObjects.InsertCopy()
+ new viewoption: wpDrawPageMarginLines to draw a doted line round the text area
+ function 'CodeListTags', works like 'GetCodeTags' but creates a TWPTextObjList
* cleaner display of text selection (visible with italic text)
* improved paragraph-border drawing (respects indent-left/right)
+ RTL support can be activated by setting the flag paprRightToLeft in TParagraph.prop
  or for all paragraphs with WPRichText1.Memo._RTLSupport := TRUE
- bug in function GetPointFromParLin removed
- improvements to automatic table header/footer feature
  a) to avoid tables without data row
  b) to work with hard pagebreaks
* improvement to HTML writer for better tables
- WPPREMIUM: fixed textbox load+save to save compatible position and size information
- WPREPORTER: "pagebreak after data band" option is now working
+ WPFORM SUPPORT: now operational in WPForm V2.50 - only text rotation is not possible
+ ALL underline modes and the underline color feature are now working
* automatic update of filter index for load/save dialogs
+ property TextLoadSaveOptions. Makes it possible to set a format string for
  Load, Save and SaveAs operations.
+ The attribute interfaces (such as WPRichText1.WritingAttr) now have a overloaded Clear procedure
  which allows it to set the fontname and -size, + color right away:
  WPRichText1.WritingAttr.Clear('Verdana',10);

7.4.2005 - V5.0 RELEASE 14

* improve import of V4 mail merge templates. In RTF reader (wpioreadrtf.pas) the $DEFINE FIXUP_V4_FIELDS
  enables that trailing >> and] are automatically removed.
- WPPremium: fixed bug in copy&paste of footnotes
+ WPPremium: cursor movement within ootnote blocks using cursor up/down
- WPPremium: fixes to footnote format routine
- WPREPORTER: fix to merge fields in second table row of a band
+ "image under text" option in graphics menu of default actions
+ cursor movement within visible header/footer blocks using cursor up/down and mouse click
+ improvements to the handling of forms (see new demo: EditFields)
- improved format routine to support better text wrap around images + text justification
* Revised border dialog. (All properties are undefined by default, can be changed individually)
* LoadFromFile, LoadFromStream now loads header/footer even if there is a body text defined
  (the function IsE,mpty) has been updated
- several fixes to stream sections in RTF and WPTOOLS format
+ the property WPAT_ProtectedPar is now also checking attached styles and parent table/rows
* copy&paste, Drag&drop now automatically tries to not create orphan field opening/close tags
+ added support for IME editor
+ the ruler now alows it to change the left indent only
- bug fix: space before + page break
- improved RTF reader: load bookmarks in table cell bug fixed.
- possibility to ignore font, font sizes when reading/pasting text
* better update of WPComboBox
- fix for WPTOOLS reader when fields params were enclosed in [ ] signs
* possibility to have a tabstop before the left indent
* new handling of font names in RTFPros object (list instead of array)
* changed logic of Ctrl+Left/Right to jump to start of word instead of end of word
+ font charset load&save in RTF format
* several improvements to cursor positioning in forms (ProtectedProp=[ppAllExceptForEditFields])
+ new RTF writer option: "-nonumberprops" - in RTF the numbers are saved as regular text
+ new writer option: "-nomergefields," - merge fields are not saved, just the embedded text
+ new writer option: "-nohyperlinks" - hyperlinks are not saved
+ new writer option: "-nobookmarks" - Bookmarks are not saved
+ new RTF reader option: "-ignorerowmerge" - ignore the combine rows RTF tag
+ procedure 'FastAppendText' is now a function. If the optional parameter 'AsNewSection' is true
  this function will return the reference to a new section property object.
* linked images are now saved with width/height (Word still ignores the \w \h parameters)
+ the following functions have been added, they now include table cell handling:

function DeleteParWithCondition(Condition: TWPCheckParagraph) : Boolean;
function DeleteParWithEmptyFields : Boolean;
function DeleteParWithText(const FindText: string) : Boolean;
function DeleteTrailingSpace(EmptyFieldsToo: Boolean): Boolean;
function DeleteLeadingSpace(EmptyFieldsToo: Boolean;InFirstPar : Boolean = TRUE): Boolean;
- ... lots of small fixes to improve overall perfomance and reliability

WPTools 5.0 - Release 13 (11.3.2005)

- solved problem which occured under windows 98 (critical fix)
+ new IDE context menu for the TWPRichText 'Change Page Size'
- RTF reader/writer now work better with paragraph styles
- bug fix for the initialization of text which uses a character style sheet
- show 'Custom Size' in page property dialog when printer sizes are loaded
+ new auto zoom mode: wpAutoZoomAsManyAsPossibleInRow
+ new demo: AppendAsSection: Create a big text out of several texts incl header + footer
+ enhanced the WPTOOLS format reader to understand the <newsection/> tag. This makes it possible to create multi section texts without loading them into an editor!
+ Enhanced API to work with sub paragraphs. Plese see the new demo "SubParagraphs"!
* InputTextField/InputTextFieldName are now functions returning the created TWPTextObj
+ new event: BeforeDropText to abort drop operation
+ property WideStringValue : WideString of 'Contents' in OnMailMergeGetText
+ new format string option: '-codepageXXXX' to set code page for ANSI reader/writer
+ property Text and SelText now use the current keyboard codepage (internally uses '-codepage' !)
+ possibility to create repeated table header and footer rows. See updated demo: CreateTable
+ possibility to use the TWPFormulaInterface to calculate the value for repeated fields in header/footer(OnTextObjectPaintCalc) See TableCalc demo.
+ WPReporter: added the possibility to have repeated header/footer with WPReporter

WPTools 5.0 - Service Release 12.2 (22.2.2005)

* improved code to size table rows (requires EditOption wpTableRowResizing)
+ possibility to set a fixed table row height in editor (press CTRL)
+ vertical alignment in table cells is now handled
+ page up/down key is now handled differently to avoid deadlock
- bugfix in function TParagraph.SplitAt (stability issue)
- bugix in text objects drawing routine
- bugfix in ClearUndoStack (stability issue)
* better handling of merged rows - now the last column can be always merged even if the column count in the rows is different
- fixed bug with inherited attributes in table cells
- check for memory and resource leaks
- fix in print routine to better handle beginprint/endprint
+ option: "-nobinary" to save RTF code with hex encoded data
+ save binary RTF variables
- fixed bug in format routine which was only visible when optimation was off
- fixed bug with SpeedReformat when paragraph at page break wrapped.
- fixed bug in reformat routine which caused wrong positioning of movable images which were too large
* improvement of scroll function: Scroll to selected object instead of anchor
* The footer now always starts at the bottom of the page with MarginFooter distance. If you need the old behavior with the footer to start at the beginning of the footer use FormatOption wpFooterMinimumDistanceToText
* started implementation of character styles
* the RTF writer interpreted OptOnlyBody incorrectly and so didn't write some RTF tags
* improvement to EditOption wpAutoDetectHyperlinks to automatically exit a link when space or ')' is typed.
- fixed problem with seletion of textobjects or codes with mouse
- fixed problem with KeepTableOnPage and movable images
- removed problems which occured when cursor was at the end of a line

WPTools 5.0 - Release 12 (3.2.2005)

+ the EditBoxModes wpemAutoSizeWidth and wpemAutoSizeHeight and the events OnChangeEditBoxWidth and OnChangeEditBoxHeight are now working. **New demo: "EditBoxModes"**
+ format option: wpfKeepOutlineWithNext to keep headline with chapter text even if seperated by empty lines
+ PrinterParameter are working now. Please check out the **new demo "PrinterSet"**
* updated chapter 'Mailmerge'
* new chapter TWPTextObj with custom draw event
* save non standard RTF-Variables as 'userprops' to RTF
+ format option: wpfKeepOutlineWithNext to keep headline with chapter text even if separated by empty lines
- paragraphs longer than 2 pages were not formatted correctly.
- better sizing + movement of images in "normal" layout mode

* sizing rectangles are not any longer shrinked when zooming out
* save and load new page starts before a table row in a Word compatible way
- fix in RTF reader to avoid problem with RTF code where style properties were written before \intbl
- fix in RTF writer to write the \intbl property correctly
- better handling of keepn
- copy from table cells - improved
- changes to indents in selected cells now works properly using the ruler

WPTools 5.0 - Release 11.1 (23.1.2005)

+ Readonly property for header/footer (TWPRTFDataBlock class) - if true they cannot be selected in page layout mode
* EditOptions spreadsheetcursormovement now works (jump to next cell with TAB)
+ TWPVirtPagePaintParam which is parameter of event: CustomLinePaintBefore
  now has new boolean property: PaintingInEditor
+ unit WPSyntaxInterface to use the SYNTAX HIGHLIGHTING modules which are
  part of SynEdit with WPTools 5. (See demo SynHighlight)
+ SPEEDREFORMAT (can be disabled in FormatOptions): During regular text editing only
  the current and the next page is formatted, the rest of the text is untouched until
  the next time CR or Ctrl+CR is pressed or the user scroll the text.
  The delay between keystrokes which was noticeable with very long texts is so minimized.
+ new component: TWPPaintEngine . Used to paint the text from a TWPRTFDataCollection.
  See demo: Tasks\DynAssignRTFData
+ KeepN support activated
- Widow/Orphan control revised (--> property FormatOptions)
* improved handling of double buffer. Now the creation of 1000+ editors at a time is possible
- two small editing bugs solved (delete selection of one char, up/down in table)
+ Hyphenation (manual, use Ctrl + '-' to mark a character to go into next line)
+ Support for BB codes in the HTML reader. Must be enabled with -useBBCodes in format string.
  Use -ignorehtml to ignore regular HTML tags
+ use #13 code to create new paragraphs in HTML. Must be enabled with -useCRin format string
- the DefaultAttr and the WritingAttr didn't survive a Clear of the text. This has been fixed.
* better line number display in WPGutter - now start with 1 instead of 0
- change to make sure the selection/cursor is displayed at once when cursor keys are pressed! (define UPDATE_CURSOR_ATONCE)
- LoadFromFile now uses the FormatString parameter
- Load&Save Section in WPTOOLS Format, too
- fixes to WPReporter

WPTools 5.0 - Release 10.5 (22.12.2004)

* faster paint routine
* new Event: OnNewRTFBlock: Makes it easy to apply a default font/style/text (see online HLP)
* TWPObject.Paint() function now receives TWPTextObj and Mode paremeter to adjust painting for different devices
- several bugfixes (see history.txt)

WPTools 5.0 - Release 10 (7.12.2004)

***** new unit WPCreateDemoText - must read chapter: Set Attributes in code
* new demo LabelPrint which implements an easy to use form to print labels.
* new demo ExternalPages to show how to mix custom printed pages witn text
* new demo Find Text
* updated demo: CreateTable
* update demo: GlobalStyle (includes self running demo)
* updated section in this manual: Header and Footer
* updated manual section and VCL functionality: Interactive Text
* far improved RTF reading and writing
* improved WPTOOLS format reader and writer
* improved HTML format reader and writer
+ support for legal outlines
+ updated CreateTableOfContents procedure
* many fixes and additions to programing API

WPTools 5.0 - Release 9 (26.10.2004)

* better handling for PageWidth/PageHeight actions
* better handling of the tables with WordWrap set to TRUE
+ added demo "ThreadSave" to show how to do threadsave mailmerge
+ added procedure: DeleteFields
+ now possible: colors and sizes for bullets
+ added FormatOption: wpfAlwaysFormatWithScreenRes - for better display when you only display on screen but do not print
+ added event: OnCustomLinePaintAfter - print borders around paragraphs or group of paragraphs
+ added event: OnCustomLinePaintBefore- print background of paragraphs or group of paragraphs

+ added event: BeforeInitializePar - syntax highlighting, dynamic hiding of pars etc
+ new: powerful TWPSuperPrint component to print labels, multiople pages on one page and booklets
+ WPReporter: added TWPFormulaInterface - calculation in tables, paragraphs and "CALC" fields
- fixed format routine: centered and right aligned text was not handled correctly when indents were used.
+ added localization

NEW and updated Demos:
* CreateTable - shows how to modify an existing table!
+ SuperPrint - print labels and booklets using a new powerful component: TWPSuperPrint
+ ThreadSave - merge letters in a several threads
+ Mini - create a compact wptools editor window with split screen in code
+ Localisation - localize wptools at runtime
+ WPReporter_Calc - requires WPTools bundle: powerful calculation in text and tables

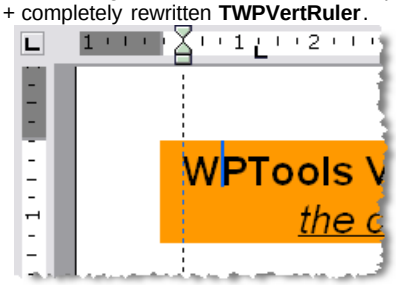
WPTools 5.0 - Release 8 (6.10.2004)
- fix to GetXPositionTw and GetYPositionTw to work with zooming and XOffset properly
+ new procedure: Memo.GetXYPositionAtRTFTW
* improved unit WPObj_Image to save compatible RTF images in BMP, WMF, JPEG and PNG format (see property WriteObjectMode. It must be set to wobRTF for this)
* improved image saving code for PNG files - they are now saved compressed in WPT format
- improved painting of bullets created with WPTools 4
+ added support for BCB6 and BCB5 (WPTools Standard Edition)
+ added events to TRTFDataCollection to modify reader and writer (BeforeSaveToStream etc)

WPTools 5.0 - Release 7.5 (30.9.2004)
+ new MDI Demo - shows how to use DefActions with MDI form
+ new "Lines" property for Editor and RTFDataBlock
* improved DefAction Module
* revised RTF saving code: style and table handling improved. Fixed problem with nested tables.
- RTFwriter: In landscape mode the non-swapped page size values were written
+ new FormatOption: XMLOutline - shows the text as HTML outline
* revised HTML reading logic
* support for hyperlinks and bookmarks when printed with wPDF
- improved drawing of tables in header or footer
* added support for EDSSPELL

WPTools 5.0 - Release 7 (13.9.2004)
+ added reusable data module WPDefAct - it contains main menu, image list and actions
+ added new popup editor WPDefEditor - it can be used in your applications (read more)
+ new procedure SetZoomMode to change layout mode and zooming quickly
+ improved performance of MergeText and FastAppendText

WPTools 5.0 - Release 6.5 (8.9.2004)
+ Redo support (activated in EditOptions!)
- bugfixes to cursor movement procedure
- improvement of formatting procedure
* added property DrawOptions to rulers
* added OnPaint event to rulers

WPTools 5.0 - Release 6.5 (4.9.2004)
+ completely rewritten **TWPRuler**
(many properties need to be ignored when the form is loaded in the IDE)
The ruler now supports undefined, inherited indents and grays out tabstops which are not active in al selected paragraphs.
The height / width is now fixed to 24 pixel. This is important for a clear display.
+ completely rewritten **TWPVertRuler**.



+ Page relative images are now supported : PositionMode **wpotPage**
+ added autoscroll (with variable speed) - can be switched off in EditOptions
* improved selection across page borders
- fixed problem in spellcheck interface
- fixed problem when saving tab stops
* added EditField demo
- several bugfixes and additions to API

WPTools 5.0 - Release 6 (24.8.2004)

- renamed unit WPOBJImage to WPOBJ_Image to avoid conflict with TWPObjType: wpobjImage
+ new Demo 'WaterM2' shows how to draw a form or image tiles in the background
* updated Demo 'ParStyles' - to also show how to work with style combo box and dialogs
* The PRO version is now compatible to BCB5 and BCB6 (see BCB notes)
- renamed unit WPOBJImage to WPOBJ_Image to avoid conflict with TWPObjType: wpobjImage
* improved deletion of selected tables rows
+ added support for references (reference: display page number of page with bookmark)
* improvement to bullet handling
+ reformat optimized for best screen and print quality (see WYSIWYG)
* WPTools format handles Numberstyles, complete style table and RTFVariables
+ improvement to function Draw() - now also work when SetWindowOrg API is active
- bug fix in SaveToFile() - parameter Format was ignored
+ procedure ParStylePaint to paint a style name into a listbox or combo
+ added: StyleDialog
+ modernized text selection with mouse (selects words automatically)
* updated drag&drop code, now also between different TWPRichText
+ added: StyleSheetDialog
* finished: TWPStyleCollection (please note: You don't need it for the paragraph style support.
It is only a container. The dialogs are attached best to an editor using property 'EditBox'
+ added style saving to RTF
- improved style loading from RTF
+ optimized WPReporter drawing code
* added popups to WPReporter Editor Dialog

WPTools 5.0 - Release 5.5 (11.8.2004)

+ new function: WPRichText.Assign - copies the text and the view options + special text attr.
* updated TWPStyleCollection. This class stores the style templates
+ new Demo 'ParStyles' - this shows how to work paragraph and span styles - natively and fast
* many improvments to HTML loading and saving, esspecially the CSS support has been updated
+ several improvements to the programming API to make it more consistent
- solved problem with tabstops
* extended TBX demo (also see Use WPTools5 with TBX)

WPTools 5.0 - Release 5.1 (6.8.2004)

+ Editor switches off numbering on Return in empty line
+ new demo: GridMode - create a table from text and images loaded from database
+ loading and saving of numbering (complete new code to write list styles)
- bugfix for property CPColNr
- better handling of TWPRichText.DefaultAttr (it was only partly used)
- fix to property 'Readonly' - was also declared also in unit WPCtrRich
- improvement to SpellAsYouGo: do not check word during writing
- fix to OnDblClick. The last parameter 'Ignore' now is passed as 'var'

WPTools 5.0 - Release 5 (2.8.2004)

* All Layout Modes are now operational - see Layoutmodes
* All ViewOptions now work (ShowCR etc)
* new table resizing code
* hyperlink support in RTF label
* added: OnPageGapGetText
- improved RTF and WPTOOLS reader and writer classes
- improved loading of RTF text which contains charsets
- improved loading of tables
- improved HTML writing: saving of <font> and tags
- improved HTML writing: saving of
 tag
* added OnChange event
- removed LayoutModes 'wplayFullLayoutColumns' and 'wpThumbNailViewNr' (redundant)

WPTools 5.0 - Release 4.5 (25.7.2004)

- improved handling of soft line breaks = Char(10)

- improved action handling
- new pseudo action: 'TWPToolsCustomEditContolAction' which is used to replace the TWPToolControl
- added TBX demo (also see Use WPTools5 with TBX)

WPTools 5.0 - Release 4 (16. July 2004)

* many improvements to RTF reader (load header/footer for sections)
- improvement to rendering of tabstops
- improvement to formatting of justified text
+ new method: HyperlinkConvertOldWPT3Links to convert the old WPTools hyperlink syntax
- switch off unwanted painting of paragraph borders
* increased performance of InputString()
+ Support for overwrite mode: new property Inserting and TextCursor.Inserting
+ added funtion GetPar(parindex) : TParagraph
+ added property ProtectedProp and event OnCheckProtection.
* improved "editfield" protection and edit code, also added edit-field events
* improvement to HTML writing code to reduce file size
- WPRuler now handles tabstops
- fixed bug which SetCharAttr used by the ChangeAttr demo

WPTools 5.0 - Release 3 (2. July 2004)

- undo for image moving + resizing
- undo for Drag&Drop and Copy&Paste
- improvement to format routine to better support text wrapping around images
* save header and footer to RTF - now also the optional names of header and footer are saved!
- improvement to selection + cursor placement
- fix to avoid unwanted drag&drop
- revised loading and saving of fields and objects from and to RTF
- **Image handling**, resizing and moving - highly improved for character and paragraph dependent images
- now supported: different wrap mode (TWPTextObj.WRAP)
- bugfix: format routine: word wrap around images did not work at start of paragraph
+ **WPReporter** 2 - beta 1 - now with new group folding function.
  Currently only 'IgnorePageHeight' operation supported
+ WPReporter: added WPEval Engine and created functions to make it possible
  with WPReporter to change text styles in scrips (= band commands)
+ WPReporter: added improved band dialog, now with insert/delete band buttons
+ added unit WPWordConv
+ added unit wpManHeadFoot
- improvements to the handling of property 'WorkOnText'
- property "ScrollBars" works as expected
- added **undo** support (70% complete)
+ new ViewOption: wpDontPaintPageFrame
+ new ViewOption: wpCenterPaintPages - to center the pages automatically in the preview dialog
+ new property for TWPPreview: SinglePageMode. If true only one row of pages are displayed (1 or 2)

WPTools 5.0 - Release 2 (17. June 2004)

- The HTML loading has been improved.
- The function Draw() is now working. Please note the new demo project FunctionDRAW. Draw will render the text using the same word wrap as it is used in the editor. It is used to fill rectangles vertically with text. A new rectangle can be started when the text was not completely printed.

WPTools 5.0 - Release 1 (14. June 2004)

This first release includes the powerful new RTF engine with its versatile capabilities to use paragraph and character attributes. This versatility does not only come from the amount of possible attributes, but how the attributes can be stored - attached to a paragraph or a style or inherited.

The GUI controls have been taken from WPTools 4 and adapted as far as possible.
The look and feel was not changed - on purpose. Later new property dialogs will be delivered.

4 Notes for Upgraders

As emphasized before WPTools 5 is based on a new RTF Engine. (We call this RTF engine although it internally works with data structures which are much more similar to HTML/CSS than to RichText-Format)

This means that if you have modified the RTF engine before or use undocumented features the affected parts of the code must be updated. We will assist you here to make the transition easier. WPTools 5 contains so many new features and possibilities that it is very likely that a very elegant way to solve the problem can be found.

Please check out the FAQ web forums
FAQ: Questions when upgrading
FAQ: General

Important:
WPTools 5 uses 'delayed reformat'. This means the formatting of the text is delayed until the next idle phase. Therefore, If you are using InputString or similar it is not required to use BeginUpdate/EndUpdate to speed up the application. On the other hand, if you want to print the text (or convert it to PDF) right after loading or creation, it it is required to execute the procedure **ReformatAll**;

If you used to create an editor at runtime using TWPRichText.Create(nil) please change the code to use the constructor CreateDynamic; (C++Users call the method _MakeDynamic() after the new) This way the editor knows it is invisible and has no parent. You need to execute ReformatAll whenever you need the text to be formatted for printing or export to PDF.
Do not create editor windows which should work interactively using this method!

Overview

When you convert older projects you will at least have to modify the uses clauses

remove: WPDefs, WPObj, WPRTF*, WPPrint, WPRich, WPWinCTR

add: WPRTEDefs, WPRTEPaint, WPCtrMemo, WPCtrRich, WPObj_Image, WPIO

Please note that code which works with pointers (lin : PTLine, pa : PTAttr) cannot be easily converted to work with WPTools 5 so please do not try to update the old logic to work with WPTools 5. It is better to find a way to do it using the new, advanced API of WPTools. Please don't hesitate to post a question in the forum: Many problems which required hundred lines of code were converted into a few lines with "WPTools 5" code.

Style handling is now done inside the RTF-Engine. The WPStyleCollection is still there but it is only a container for template styles.

WPReporter works differently in V5 - please see WPReporter

4.1 Changed Unit Names

One of the first things you might realize when you try to compile a WPTools V4 project with WPTools Version 5, is that some unit names no longer exist.
This is because we decided to create new units for program code which is entirely new. At the same time obsolete units were deleted.

As a result the following changes are required:

Change
WPDefs, WPObj, WPRtfTXT, WPRTFIO
all to **WP*RTE*Defs**

Change
WPRtfIO, WPRtfInp
to **WP*RTE*Paint**

WPRTEDefs and WPRTEPaint are the main units of the "RTF-Engine". Their pascal code is only included in WPTools Professional and WPTools PREMIUM.

Change
WPPrint, WPWinCtr
to **WP*CTR*Memo**

The unit WPCTRMemo contains the editor control.

Change
WPRich
to **WP*CTR*Rich**

The unit WPCtrRich contains the procedures, objects and properties which were mainly added to provide compatibility to older WPTools versions. It is required by the TWPToolBar, TWPRuler and TWPAction classes.

The main WPReporter class TWPSuperMerge is now located in WPRTEReport, not wpmerge.pas.

4.2 Changed Classes

RTF-Engine

The RTF Engine concept has been changed greatly. It is no longer possible to use a RTF-Engine object, such as "TWPRtfTextPaint" in WPTools 4, to identify certain text or text properties - In WPTools 5 the text objects are accessed directly, not through the 'RTF-Engine': The TWPRTFProps are now used for text properties, such as colors and styles. For text either the TWPRTFDataCollection or the TWPRTFDataBlock is used. An exception of this rule is the cursor class, also accessible through TWPRichText.TextCursor. It contains many variables and procedures which used to be in the TWPRtfTextPaint object.(see Data Structures)

Usually code does not require a reference to the RTF-Enigne since this reference is provided by the editor control (TWPCustomRichText, TWPCustomRtfEdit).

Thus, in general it is possible to replace something like "RtfText : TWPRtfTextPaint" with a reference to a TWPCustomRtfEdit.

Editor

The TWPCustomRtfEdit class is defined in the unit WPCtrMemo. It contains access to the RTF engine (Memo), the TWPRTFData object (Memo.RTFData) and the possibility to change attributes (through Memo.Cursor). However, it does not contain the 'CurrAttr' property, which has been defined under the unit WPCTRRich, in the class TWPCustomRichText.

4.3 Changed Pointers

To make compilation with Delphi8 for .NET possible, WPTools Version 5 no longer makes use of pointers (exception: some optimized conversion code).

As a result, all pointers have been replaced. The PTParagraph pointers have all been changed to references to a TParagraph object. Since references and pointers can be used in a very similar manner, you can compile your code by simply removing the **^** sign (Delphi) or changing **->** into **.** (c++)

The RTF-Engine paragraph pointers (WPRichText1.**Memo.active_paragraph** and others) have been moved. The references which represent the cursor position are now in **WPRichText.Memo.Cursor** or, if no editor is used, in RTFDataCollection.Cursor.

Please note that in WPTools Version 5 the reference active_paragraph can be undefined (nil). The same is true for the 'FirstPar' reference of a TWPRTFDataBlock - although the code will create a first paragraph as soon as this property is read.

The pointer for the TLine record of WPTools 5 is no longer supported. The TLine record has been completely replaced, a position in a paragraph is now described solely by the position, posinpar.

The cursor object still supports a property called active_line. It is an integer value which can be used to access the wrapped lines.

Pointers to border descriptions (TBorder) can no longer be used. The border definitions are now stored in the regular properties of a TParagraph or TWPTextStyle class. But to provide compatibility to WPTools 4 it is possible to fill a TBorder record 'on the fly' with values calculated from the stored properties.

This means that if you have a procedure which expects a TBorder record or pointer it is necessary to pass a reference to a TWPTextStyle class and calculate the border record inside of this procedure using the TWPTextStyle procedure AGetBorder(). Please see the next chapter for further information.

4.4 Changed GUI elements

In WPTools 4 the control TWPToolCtrol () was used to create the link between a TWPComboBox and the editor.

This class is not supported anymore since it was not compatible to modern 3rd party toolbar controls. This means that it is not possible to use any TPanel as a toolbar by simply dropping a TWPToolCtrl on its surface. We recommend to use the TWPToolPanel instead.

The best approach is to use the wptools standard actions to link to buttons and menu items and the new **TWPToolsCustomEditContolAction.** The latter class can be also added to any TActionList and is used to create a link to the TWPComboBox class. (property AttachedControl)

Please read more about actions and Use WPTools5 with TBX (Toolbar2000 Extension)

4.5 New Border Handling

Border definitions are now stored in the regular properties of a TParagraph or TWPTextStyle class. These properties are usually read by the AGet function and written using the ASet procedure.

For your convenience all border properties can be saved in and restored from a TBorder record.

This record is written by the ASetBorder procedure.

The TBorder record is similar to the one used by WPTools 4, but has been improved to hold different modes, colors and widths for each line by using arrays for the values BorderType, BorderColor and BorderWidth.

Notes:
a) The blDouble and blDot LineType no longer exist. Instead the 'type' must be used to choose a different border line style.

So code such as

```
Ndouble.Checked  := blDouble in Border.LineType;
```

must be changed to

```
Ndouble.Checked := Border.AllBorderType = WPBRD_DOUBLE;
```

Please note that this is using the AllBorderType item which contains the type of all borders, if they are the same. To read or set a single style for one of the possible borders (left, right ...) use the array Border.BorderType[BlLeft..BLBar].

b) The value 'FUse256Colors' or 'Use256Colors' no longer exists. WPTools always uses at least 256 colors.

c) The elements HColor, HColorR, VColor, VColorB have been replaced by the array BorderColor: array[BlLeft..BLBar] of Integer;

So please change code such as

```
btnLColor.Tag    := Border.HColor;
btnRColor.Tag    := Border.HColorR;
btnTColor.Tag    := Border.VColor;
btnBColor.Tag    := Border.VColorB;
```

to

```
btnLColor.Tag    := Border.BorderColor[blLeft];
btnRColor.Tag    := Border.BorderColor[blRight];
btnTColor.Tag    := Border.BorderColor[blTop];
btnBColor.Tag    := Border.BorderColor[blBottom];
```

d) The Thickness value no longer exists. Please use either the AllBorderWidth or the BorderWidth[] elements. Please note that the width is stored as twip value, not as 1/2 pt.

e) The Space value is no longer used. The styles now support padding values which are defined by the WPAT_ codes WPAT_PaddingLeft, WPAT_PaddingRight, WPAT_PaddingTop and WPAT_PaddingBottom. These values cannot be set in the TBorder record.

f) The function Memo.Set_ParBorder no longer exists. Instead you can use

```
CurrAttr.SetBorders(
    LineSelection: TBorderType = [blLeft,blTop,blRight,blBottom];
    WPBRD_mode: Integer = -1;
    ThicknessTW : Integer = -1;
    LeftColor   : Integer = -1;
    RightColor  : Integer = -1;
    TopColor    : Integer = -1;
```

```
BottomColor : Integer = -1;
       AllPadding  : Integer = -1;
       DeleteDefaultSettings: Boolean = TRUE).
```

A value of -1 is used to select the default value. By default using -1 will delete the corresponding attribute from the list of attributes which causes the inherited values to be used.

4.6 Changed Style Handling

In WPTools 4 the style handling was performed by the TWPStyleCollection component. This is not the case anymore.

This component can be optionally used to store the style in their non-binary representation (which is a string list with name, value pairs) and assign this style to one or more TWPRichText when required. This can help to synchronize the style sheet to provide a default sheet.

The development of the new TWPStyleCollection has not been fully completed so we suggest not to use it right now.

Please read here about how to use the paragraph styles in a native way.

4.7 BackgroundImage

WPTools 5 does not have the property "background image" anymore.

Instead you can draw the image using the water mark event - see Demo 'WaterM2'. You can create a different tiled background on each page and it is also possible to show a form in the background. WPTools 5 takes care about the necessary buffering so no flickering will be visible.

We recommend to use RTFVariables to store the file name of the background image which should be used with a document.

5 Guide

In this chapter we collected introduction texts to important tasks which can be solved with WPTools.

We recommend to also review the HLP file since it contains a structured list of all classes, properties and methods. It also contains 'Categories' which are a big help.

In chapter 'Tasks' we will explain how to create the first small editor, how to use mail merge, how attach a generic toolbar using actions and much more. There is also an intorduction to WPReporter.

Please check out the demos which are described in this chapter:

Mini Editor - shows how to create a small, but powerful editor
Create Table in Code - demos the most effective way to create tables in code
TBX Demo - how to use WPTools with TBX, the toolbar 2000 extension
GridMode - how to create tables dynamically from a database
ParStyle - this demo shows how to work with the CSS like paragraph styles
Localization - how to localize the dialogs and messages.

5.1 Data Structures

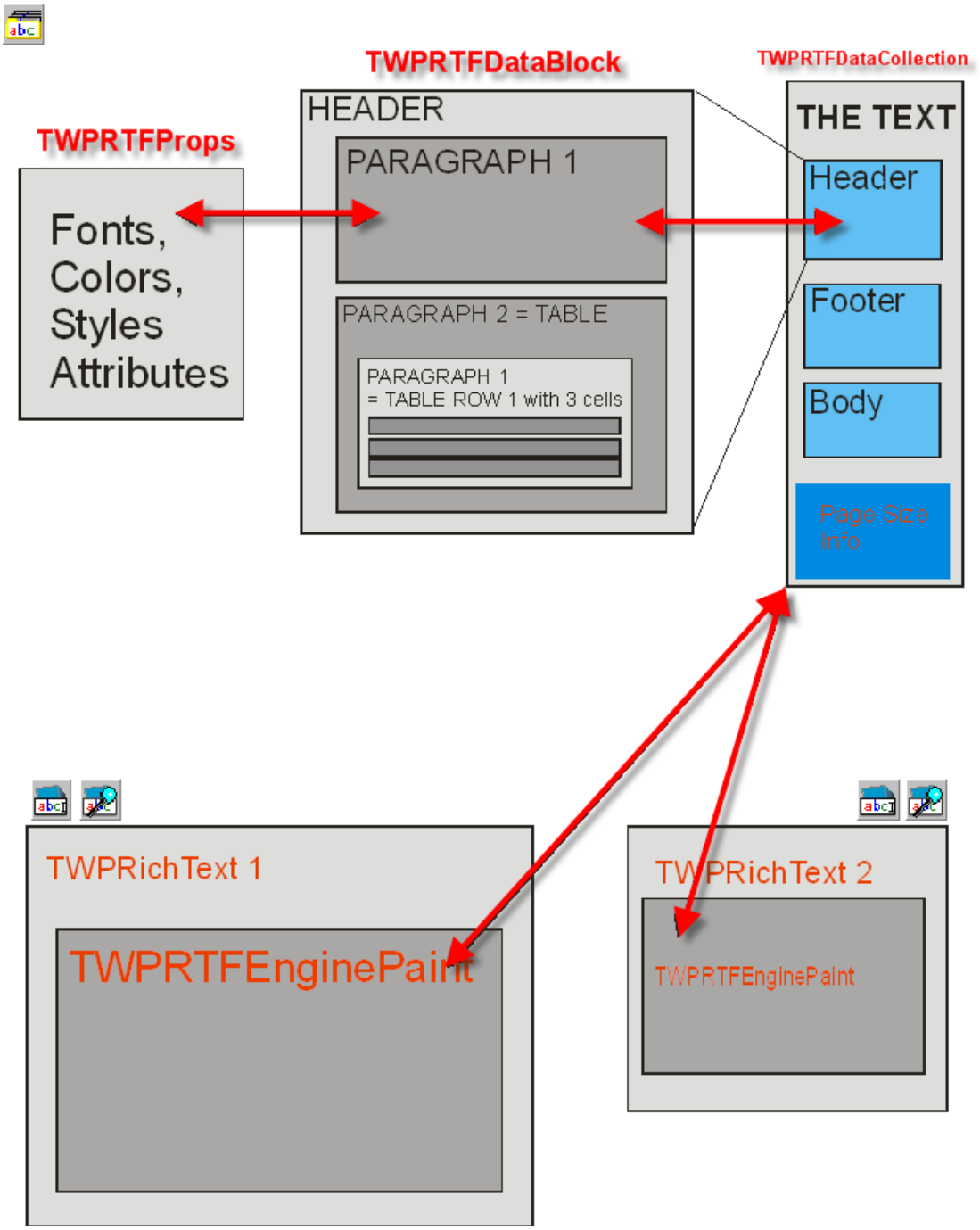
Here is a general description of the architecture and concept of the RTF engine, for your reference.

Please read this carefully!
This knowledge is necessary for you to understand WPTools 5.

Since not all details are listed here, please look for further information on the classes written in bold in the online help.

In the WPTools RTF-Engine (we always refer to the 'RTF-Engine', however this does not mean that the engine is limited to the Rich-Text, *.RTF), a text is split up into several parts. The main parts are stored in two objects which are linked together:

a) RTF Data is stored in the **TWPRTFDataCollection**
b) RTF properties, such as paragraph or number styles, are stored in the **TWPRTFProps** object.



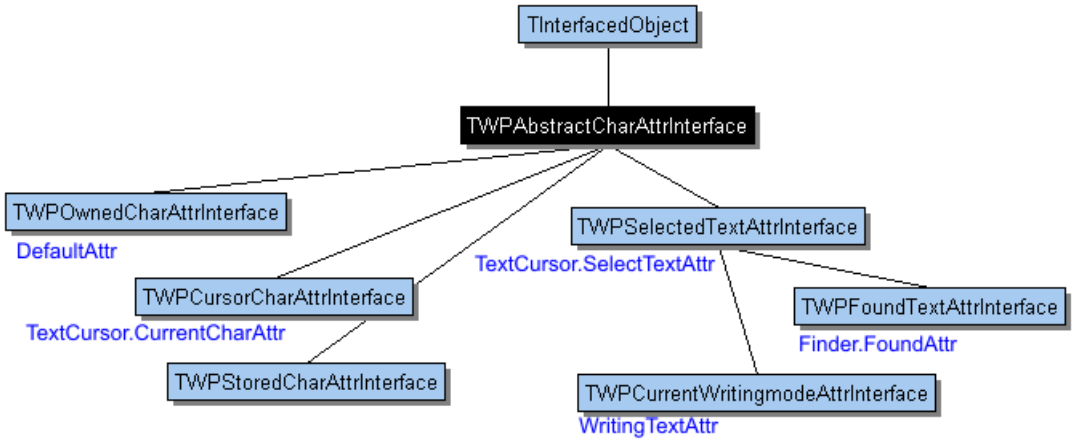
Note: This concept allows multiple TWPRTFDataCollection objects to share the same TWPRTFProps object. Thus, they share the same attribute identifier (such as index values for colors). If you use this feature you can simply copy texts parts between RTFData objects or compare text.

The TWPRTFDataCollection also hosts the text cursor (**TWPRTFDataCursor**) and a few parameters which are shared by the RTF editors (TWPRTFDataCollectionEngineParams). This means that even if you have several editors using one TWPRTFDataCollection there is only one cursor which is the same for all editors attached.

The cursor object also controls text selection and changing properties of the selected text

(SelectedTextAttr : **TWPSelectedTextAttrInterface**) or the current writing mode (CurrentCharAttr : **TWPCursorCharAttrInterface**). It also contains the CPAttr (**TWPTAttrEmulator**) interface which changes the attribute at the cursor position.

All classes which change the attribute of certain elements inherit from TWPAbstractCharAttrInterface. In cases where it makes sense the classes are also able to change paragraph attributes as well. The only exception is TWPAttrEmulator, which does not work like the other interfaces since it is mainly used to offer compatibility to the WPRichText.CPAttr pointer in WPTools 4.

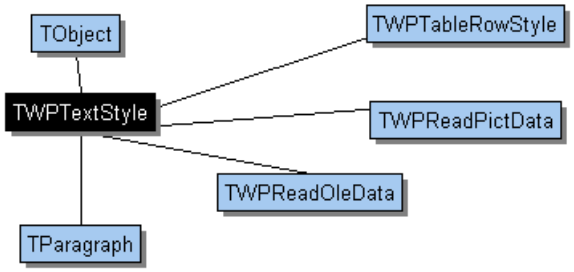


The TWPRTFDataCollection is home to the **TWPRTFDataBlock** collection items. Such an item contains the text which is displayed. The text body and the header or footer texts are all different collection items. When a new text is loaded, it is first loaded into a new TWPRtfDataBlock and, if everything is all right, then inserted into the body. The editor can display any of the RTFDataBlocks, or even display several at once.

The TWPRTFDataBlock contains the text within a nested list of TParagraph objects. The **TParagraph** objects are linked using the references NextPar/PrevPar and for nested dependencies, ChildPar/ParentPar references.

Note:
WPTools 4 only supported linking in one level using next/prev pointers. The new TParagraph object contains functions to emulate these pointers. Using this function it is still very easy to create a loop which checks all paragraphs in a text. The first paragraph is referenced by the property FirstPar.

The TParagraph class inherits the complete functionality of the **TWPTextStyle** class which contains the code to maintain attributes and tab stops. The TWPTextStyle class is also used by other classes which need this functionality.



How does TParagraph store the text?

The text is separated into characters, character attributes and objects. Each of these elements is stored in its own dynamic array. For the characters an array of WideChar is used, the character attributes are stored in an array of cardinal (double word) values. When objects are used, you can read the TWPTextObject for a certain position in the paragraph using the array ObjectRef. The count of elements is stored in the variable CharCount.

Note:
The TParagraph class has several functions to insert and delete text and objects.

In the instance that it is part of a table, TParagraph also has functions to find other parts of the same table (rows, cells or the parent table object). The memory architecture of a table is very similar to the system used by HTML:

(All rows of a table and all cells of one row are connected using NextPar/PrevPar, the levels are created using ChildPar/ParentPar.)

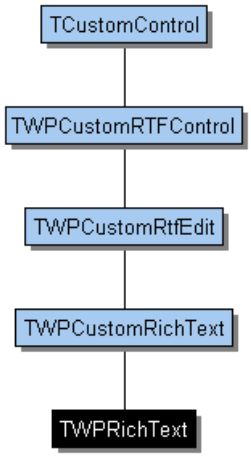
There are also useful properties which provide reference to the parent row or the parent table of a cell, or, for cells which are in a nested table (= table in a table cell), to get the first level ("ParentParent") row or table.

As stated character attributes are stored in just one double byte value. You may ask, 'How can this work?' Particularly since WPTools 5 supports 15 different character attributes with multiple settings possible for each of these.

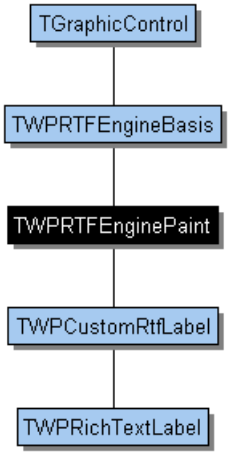
WPTools 5 does not save the attributes directly in this CharAttr value. It only saves an index there. This is then used to retrieve the actual attributes from a global attribute cache.
We find this concept ideal - other text editors use start/end tags to store character attributes, others even split up the text into elements which are using the same combination of attribute styles. In both cases it is extremely difficult to 'apply' a certain attribute to text. Our concept makes it possible to simply set a number value and the style is changed.
If styles have to be updated, the interface classes (TWPAbstractCharAttrInterface) mentioned allow a new value to be created from the existing settings and the new one.

Inheritance Charts:

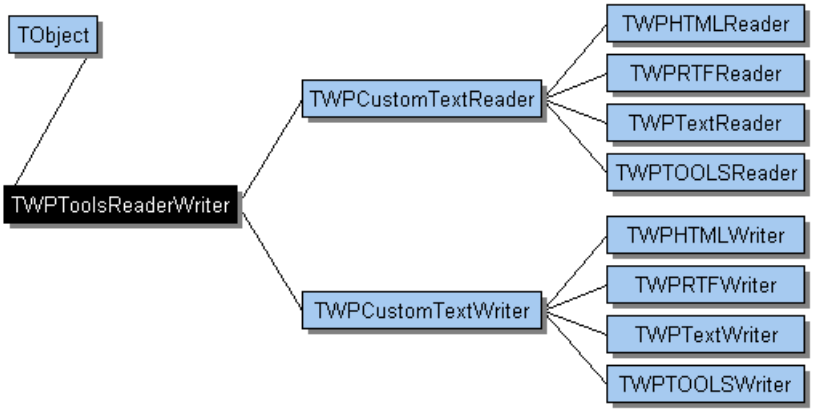
TWPRichText, the RTF memo:



TWPRTFEnginePaint, the RTF Engine (used by the TWPRichText as object 'Memo')



The reader and writer classes:



5.2 WYSIWYG

WYSIYWG - what is this ?

This word abbreviates **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et - which means that the printed output you *get* will match the screen output you *saw* before.

WPTools 5 will always work in WYSIWYG mode, this means the printed output will always match the output you saw in the editor. Making this work is actually a quite complicated task and the editing engine has to be well prepared for it. The concept of WPTools5 was created from ground up to allow several WYSIWG *modes*:

(a) Default: render for best screen and best printing quality
(b) render for optimal printing quality
(c) render for optimal screen quality - print quality can be low

Usually you do not have to change anything - but we recommend to add a switch to the application to activate mode (b) - simply execute TWPRichText1.Memo.RTFData.UpdateFormatMode(true).

To work with (c) you need to access the global WPToolsEnviroment object and change the property ScreenResolution to 0: **(GlobalWPToolsCustomEnviroment as TWPToolsEnviroment).ScreenResolution := 0;** You can also deactivate the define RM600 in unit WPCTRMemo to deactivate the improved format code for all your WPTools projects.

Please also see the chapter about LayoutModes.

Upgrade note: There are no properties ScreenResMode and WYSIWYG in WPTools 5, they are not required anymore. Using the mode (a) mentioned above provides good print out quality without the need for a default printer - this solves the problems which used to occur in applications when not printer was available.

5.3 Tasks

5.3.1 Mini Editor

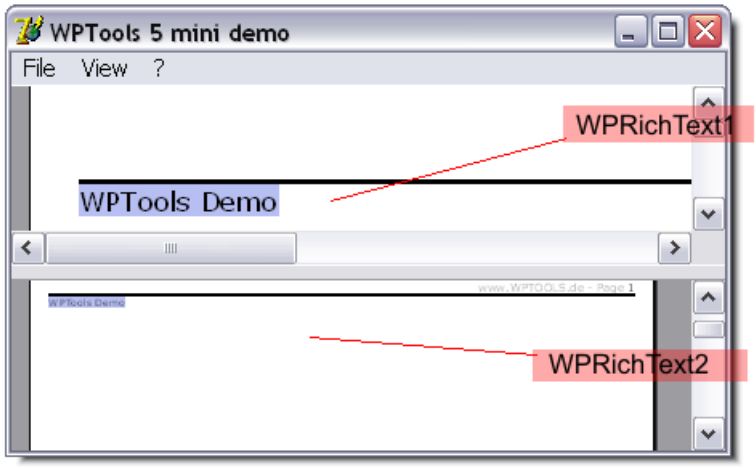
With WPTools 5 you can create extremely compact editor applications which still do not miss any functionality the end user might expect - such as support for headers and footers, WYSIWYG, scaling, tables etc.

When you need a really compact editor we suggest to create objects of the class TWPCustomRtfEdit (defined in unit WPCTRMemo) in code. This way you can also attach two editors to the same TWPRTFDataCollection to create an editor with split screen support.

Our demo uses 4 variables defined inside the form interface:

```
TWPMiniEd = class(TForm)
...
public
  WPRichText1, WPRichText2 : TWPCustomRtfEdit;
  RTFData : TWPRTFDataCollection;
  RTFDataProps : TWPRTFProps;
end;
```

We also added a splitter, a graphic popup menu and a main menu. At runtime the demo looks like this screen shot:



The editor objects are created and connected in code inside of the OnCreate event of the form:

```
procedure TWPMiniEd.FormCreate(Sender: TObject);
begin
    RTFData := TWPRTFDataCollection.Create(TWPRTFDataBlock);
    RTFDataProps := TWPRTFProps.Create;
    RTFData.RTFProps := RTFDataProps;

    WPRichText1 := TWPCustomRtfEdit.Create(Self);
    WPRichText1.Parent := Self;
    WPRichText1.Align := alClient;
    WPRichText1.TabStop := FALSE;
    WPRichText1.AcceptFiles := TRUE;
    WPRichText1.Memo.SetRTFDataOrProps(RTFData, nil);

    WPRichText2 := TWPCustomRtfEdit.Create(Self);
    WPRichText2.Memo.SetRTFDataOrProps(RTFData, nil);
    WPRichText2.Parent := Self;
    WPRichText2.Align := alBottom;
    WPRichText2.Height := 100;
    Splitter1.Top := 0;
```

We now assign the graphic popup menu. This menu is automatically used as contect menu for image objects.

```
    WPRichText1.GraphicPopupMenu := GraphicPopupMenu;
    WPRichText2.GraphicPopupMenu := GraphicPopupMenu;
```

Now we add the text to the body. The HTML format makes it easy to add some formatting.

```
    WPRichText1.AsString := '<div align=left><font face="verdana" size=2>WPTools
Demo</font></div>';
```

We also want to show a header text with page numbering. We also use HTML with the WPTools addition tag <pagenr/>.

```
    WPRichText1.HeaderFooter.Get(wpIsHeader,
      wpraOnAllPages).RTFText.AsString :=
       '<div align=right><font face="verdana">www.WPTOOLS.de - Page
<pagenr/></font></div><hr>';

    WPRichText2.SetZoomMode(-20);
end;
```

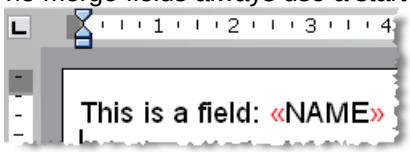
At last we use the SetZoomMode procedure to select a certain layout and zoom mode. The SetZoomMode makes it easy to modify several properties of the editor all at once. Since it only requires one integer parameter it can be used in menus or actions which are using the 'Tag' property to store the parameter for SetZoomMode.

Hint: The TWPCustomRtfEdit can also be used in a thread save context, for example to create documents using mailmerge in a thread save context. Please check out the demo "ThreadSave"

5.3.2 Mailmerge and forms

The mail merge feature is exceptionally strong with WPTools. You can merge in standard ANSI text, formatted text and images. Formatted text may be encoded in HTML, RTF or the WPTOOLS format.

The merge fields always use a start and an end marker:



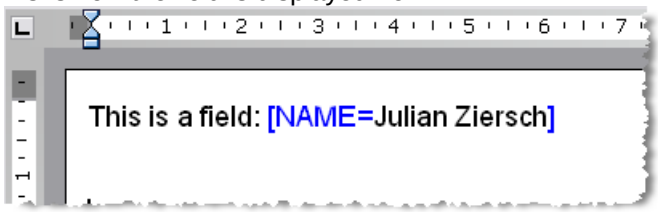
The markers use internally the TWPTextObj class. The start marker stores the name of the field in the 'name' property. The ObjType property of both, the start and the end marker is both set to wpobjMergeField.

The display of the markers is optional, it can be disabled by setting property WPRichText.InsertPointTextAttr.Hidden to true. It is also possible to show a different text in a different color.

Example:

```
WPRichText1.InsertPointAttr.CodeOpeningText  := '[%N=';
// %N inserts the TWPTextObj.Name property
// %S inserts the TWPTextObj.Source property
// %Y inserts the TWPTextObj.StyleName property - only useful for span styles
// %P inserts the TWPTextObj.Params property
WPRichText1.InsertPointAttr.CodeClosingText  := ']';
WPRichText1.InsertPointAttr.CodeTextColor  := clBlue;
```

This is now the field is displayed now:



WPRichText.InsertPointTextAttr.Hidden should be set to true when the document is printed. To delete the fields (and keep the text) use the procedure.DeleteFields.

Hint: Please check out the ThreadSave demo to learn how to use mail merge in a thread.

5.3.2.1 Create Field

To create a field use the procedure

InputMergeField

WPTools 5 also supports 'form fields'. They work like merge fields but if you use a special mode in property ProtectedProp the complete text can be protected, only the text inside this form fields can be edited. To create such a filed use **InputEditField**.

5.3.2.2 Fill Field

To use mail merge you need to add an event to the event handler **OnMailMergeGetText**.

The event receives the field name as parameter 'inspname'.

In the simplest case the event handler can be

```
procedure TForm1.DoMailMergGetTexte(
   Sender: TObject; const inspname: string;
   Contents: TWPMMInsertTextContents);
begin
  Contents.StringValue := 'This is a test';
end;
```

It is possible, for example, to use this field name to retrieve the text from a data set:

```
Contents.StringValue := DataSet.FieldByName(inspname).AsString;
```

You can also change the attribute of the inserted text using 'MergeAttr'.

```
Contents.MergeAttr.SetColor(clRed);
```

If you need to insert formatted text you can do so by assigning HTML or RTF to the property StringValue. In case the HTML or RTF cannot auto detected you can give WPTools a hint it using the flags mmMergeAsRTF, mmMergeAsHTML, and mmMergeAsWPTOOLS in Contents.Options.

To insert a picture you can use this code:

```
procedure TForm1.WPRichText1MailMergeGetText(Sender: TObject;
  const inspname: string; Contents: TWPMMInsertTextContents);
var
   img : TWPObject;
   picname : string;
begin
   picname := 'c:\sampel.jpg');
   img := WPLoadObjectFromFile(WPRichText1,picname,true);
   if img <> nil then
          begin
          // img.WidthTw := ...
          // img.HeightTw := ...
          contents.obj:=img;
   end;
end;
```

In WPTools 5 the selection and cursor is always moved o the current field. So it is also possible to **modify the text at the cursor position** directly in the OnMailMergeGetText event. This was strictly forbidden in WPTools 4, in V5 it can be used carefully:

Insert a picture:

```
  img := TWPOImage.Create(WPRichText1);
  img.LoadFromFile(picname);
  WPRichText1.TextObjects.Insert(img)
```

Insert a table:

```
  WPRichText1.ClearSelection(true);
  WPRichText1.AddTable(2,2,true);
```

Insert formatted text

```
// Delete the current contents of the field
WPRichText1.ClearSelection(true);
// Get the field default character attributes in 'WritingMode'
WPRichText1.WritingAttr.CharAttr :=  Contents.MergeAttr.CharAttr;
// And add text with variations of the attributes
WPRichText1.WritingAttr.IncludeStyle(afsUnderline);
WPRichText1.WritingAttr.IncludeStyle(afsBold);
WPRichText1.InputString('81541');
WPRichText1.WritingAttr.ExcludeStyle(afsUnderline);
WPRichText1.InputString(' Munich');
```

5.3.2.3 Create text with multiple letters

Sometimes you need to create a longer text which contains copies of the same template filled with different data.

You can use a second TWPRichText object "AllRTFText" to receive the text of all the single letters.

Then You can use code like this to loop through the complete database, merge each record and append the result to *AllRTFText*. The first time the text is copied using "AsString" to make sure the page format and the header and footers are copied too. Later records use FastAppendText.

```
var i : Integer;

Table1.DisableControls;
try
Table1.First;
i := 0;
AllRTFText.BeginUpdate;
AllRTFText.Clear;
while not Table1.EOF do
begin
   WPRichText1.MergeText;

   if i=0 then // FIRST RUN
   begin
      AllRTFText.AsString := WPRichText1.AsString;
      AllRTFText.CPPosition := MaxInt; // to end
   end
   else if i>0 then // SUBSEQUENT RUNS
   begin
      // Append a new paragragraph IF  the last line is not empty
      if AllRTFText.ActivePosInPar>0 then
          AllRTFText.InputString(#13);
      // Need page break
      AllRTFText.FastSetPageBreak(true,  true);
      // and append the text
      AllRTFText.FastAppendText(WPRichText1,false);
   end;
   Application.ProcessMessages;
   Table1.Next;
   inc(i);
end;
finally
  AllRTFText.EndUpdate;
  Table1.EnableControls;
end;
```

The above code simply copies the text to the destination. It will be one large text without sections. To create sections please add the marked lines in red. We use the section property wpsec_ResetOutlineNums to make sure each section uses its own outline numbering.

```
var Section : TWPRTFSectionProps;
```

```
try
   Table1.First;
...
         // Need page break
         AllRTFText.FastSetPageBreak(true,  true);
         // and append the text
         Section := AllRTFText.FastAppendText(WPRichText1,true);
         Section.Select := [wpsec_ResetOutlineNums];
      end;
...
   end;
   finally
...
   end;
```

When you save the resulting text you can use the following writer options in the format string to create better RTF code:

-nonumberprops: Write the numbers as text
-nomergefields: Do not save the fields, only the contents

5.3.2.4 Hide empty paragraphs

After the mail merge procedure it is possible that some paragraphs are completely empty, except for the remaining fields and maybe space and tab chars.

a) Using the function
DeleteParWithEmptyFields
this paragraphs can be deleted. This function internally uses the function DeleteParWithCondition.

The function DeleteParWithCondition works only with paragraphs and tables in the first level (not nested tables). Tables rows which do only contain cells with can be deleted will be deleted as well. If all rows in a table have been deleted that table will be deleted, too.
Inside a table cell all child paragraphs of that cell (if any) are individually checked. If those child paragraphs trigger the 'condition' to true, they will be deleted. If the main cell paragraph trigger the condition to true it will be cleared. The cell will only be deleted if all sibling cells have to be deleted, too. (the complete row)

b) You can also **temporarily hide** paragraphs which do only contain empty fields and spaces.

To do so use the event **BeforeInitializePar** with this code:

```
  if par.HasObjects(false,[wpobjMergeField]) and
     not par.IsNonSpace([wpobjMergeField]) then
        include(par.prop, paprHidden)
  else  exclude(par.prop, paprHidden);
```

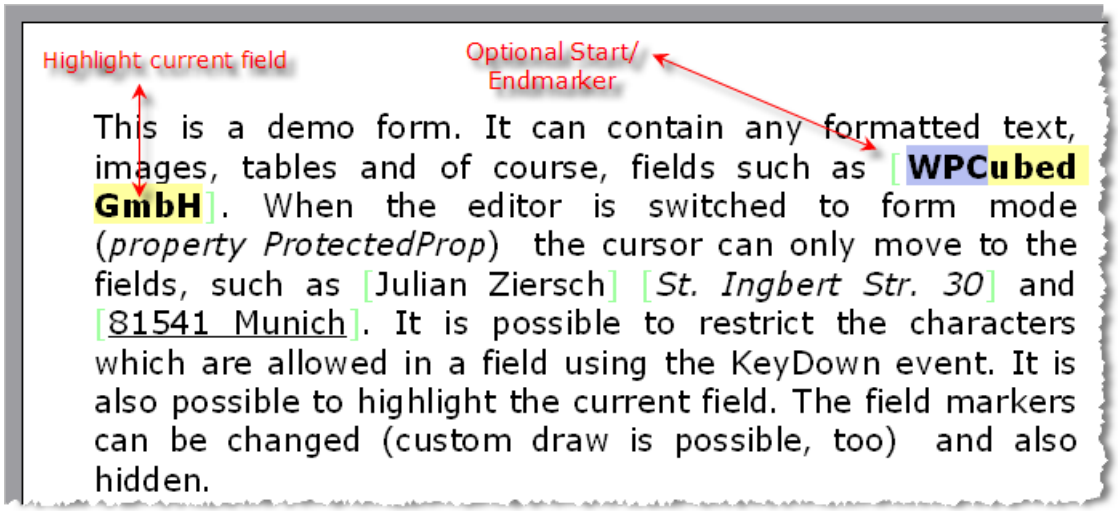
c) Delete leading or trailing spaces

```
   function DeleteLeadingSpace(EmptyFieldsToo: Boolean;  InFirstPar : Boolean = TRUE):
Boolean;
   function DeleteTrailingSpace(EmptyFieldsToo: Boolean): Boolean;
```

Both functions will stop at the first table they find. They will delete the text (and optionally empty fields) from the start or from the end of the text.

5.3.2.5 Forms/Edit Fields

WPTools can be also used to create forms. These are sepecial texts which are generally protected. The user may only edit the text in specially marked areas. We call this areas 'Edit Fields':



Edit fields work like mail merge fields. The only difference is that the objects use the mode flag "wpobjWithinEditable". When saved to RTF a \formfield instead of a \field tag is written.

To create a field use procedure

```
InputEditField(const FieldName: string;
      DisplayText: string = '';
      PlaceCaret: Boolean = FALSE;
      Command: string = ''; // will be written to TWPTextObj.Source
      Format: Integer = 0  // will be written to TWPTextObj.IParam
): TWPTextObj;
```

All procedures which work with mail merge fields will work for the edit fields, too. Like with mail merge fields the presentation of the start and end markers is controlled by the property InsertPointAttr. The text within the markers is controlled by AutomaticTextAttr.

The editor for the example above had been set up with:

```
DataEdit.ProtectedProp := [ppAllExceptForEditFields];
DataEdit.EditOptionsEx := [wpTABMovesToNextEditField,wpRepaintOnFieldMove];
DataEdit.InsertPointAttr.Hidden := FALSE;
DataEdit.InsertPointAttr.CodeTextColor := $E0FFE0;
DataEdit.InsertPointAttr.CodeOpeningText := '[';
DataEdit.InsertPointAttr.CodeClosingText := ']';
```

The most important property change is ProtectedProp := [ppAllExceptForEditFields]. This makes it impossible for the cursor to move anywhere else than within edit field tags.

To make it possible to highlight the current field (yellow background) the property UseOnGetAttrColorEvent and the event OnGetAttributeColor has been used:

```
DataEdit.AutomaticTextAttr.UseOnGetAttrColorEvent := TRUE;

procedure TWPEdTest.DataEditGetAttributeColor(Sender: TObject;
 var CharStyle: TCharacterAttr; par: TParagraph; posinpar: Integer);
var obj : TWPTextObj;
begin
 obj := DataEdit.CodeInsideOf(par,posinpar,wpobjMergeField);
```

```
if obj = DataEdit.FieldAtCP then
  begin
     CharStyle.BackgroundColor := $A0FFFF;
     CharStyle.UseBackgroundColor := TRUE;
  end;
 end;
```

Edit fields can be read out using procedure MailMerge and event OnMailMergeGetText:

```
// Read the data which is currently displayed in the editor
procedure TWPEdTest.ReadData;
begin
   FReadingData := TRUE; // global boolean to change behaviour
   DataEdit.MergeText;
end;

// Write back the data
procedure TWPEdTest.WriteData;
begin
   FReadingData := FALSE;
   DataEdit.MergeText;
end;

// This reads and writes the data fro the database 'Table1'
procedure TWPEdTest.DataEditMailMergeGetText(Sender: TObject;
  const inspname: String; Contents: TWPMMInsertTextContents);
begin
     if FReadingData then
           Table1.FieldByName(inspname).AsString := Contents.OldText
     else  Contents.StringValue := Table1.FieldByName(inspname).AsString;
  end;
end;
```

This code can be used to move to a certain field. If the cusror is within a field with that name the next field will be located and selected.

```
procedure TWPEdTest.MoveToField(fieldname : String);
begin
   fieldname := SelectField2.Text;
   // If this is the current move on ...
   if DataEdit.CurrentEditField=fieldname then
        DataEdit.MoveToNextField(false);
   // Try from here
   if DataEdit.MoveToField(fieldname,false) then
      DataEdit.SelectFieldAtCP(false, true)
   // or from start
   else if DataEdit.MoveToField(fieldname,true) then
      DataEdit.SelectFieldAtCP(false, true);
   DataEdit.SetFocus;
end;
```

5.3.3 How to use the "Default Editor"

Can you create a full blown word processing application in one minute?

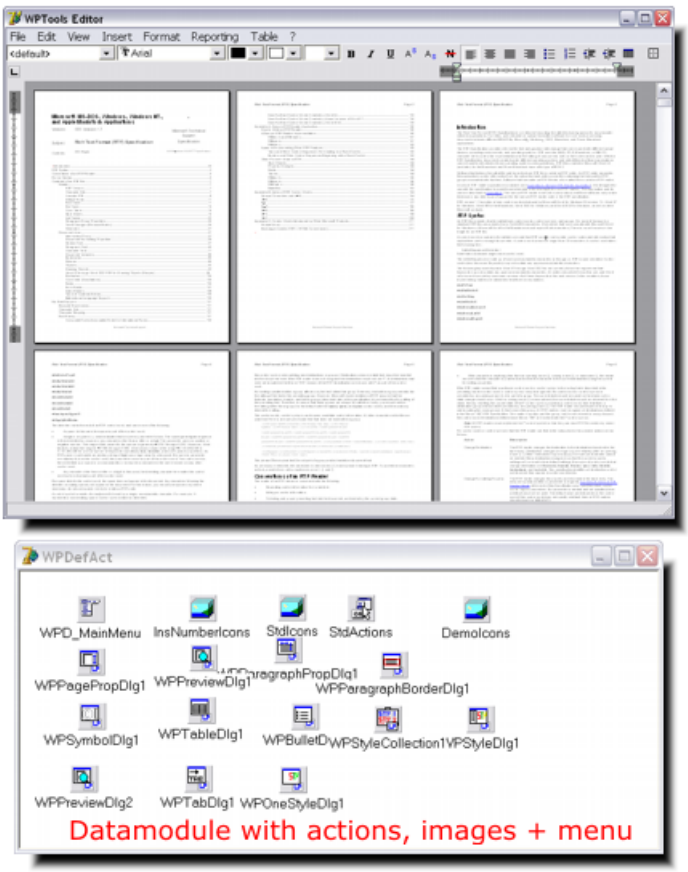
Yes, with WPTools 5 you can.

A) The is the complete program code in Delphi:

```
program UseDefEditor;

uses
  Forms, WPDefEditor;

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TWPToolsEditor,WPToolsEditor);
  Application.Run;
end.
```

All you need is to use the included default editor, this is the editor which is also used by the IDE to edit the text which is contained in WPTools objects.
It uses a form (defined in unit WPDefEditor) and a data module which contains the main menu and the actions. Of course, you can edit both files but you can modify them at runtime.
The data module with the actions will be also a great help if you need to create an editor inplace, not using the provided default editor form!

Please copy both units and accompanied DFM files to a save place if you intend to modify them. Otherwise they will be replaced by the WPTools setup.



B) Even easier to use is the component **TWPDefaultActions**. Place this component on your form - also create a toolbar (using the TWPToolPanel - you can use Copy&Paste from unit wpDefEditor), and add a TWPRuler. Now you only have to create a link to the TWPRichText in the ControlledMemos collection of the TWPDefaultActions component.
Using the OnInit event of the TWPDefaultActions component you can modify the menu.
Please see the HLP file (=reference) for more information.

5.3.4 Work with Actions

Actions objects are uses to connect menu items and buttons to a certain procedure of the TWPRichText editor.

The TActionList which contains the action objects must be specified in the ActionList property of the TWPRichText component.

To add an action to a ActionList please open its editor (double click) and select 'new standard action'

Tip: In case you find out that certain hot keys do not work please check the short cuts defined in action lists or menus. It is possible that a shortcut is consumed somewhere else.

Assigning an action to a menu item always overrides the image index of this element. So please specify the image index values in the actions and not in the menu items. The image list must be assigned to both, the action list and the toolbar!

Work with TWPComboBox

The TWPToolCtrl component is not included in WPTools 5. If you want to use the TWPComboBox control which can display a list of fonts, colors or styles, You need to create a links using the **TWPToolsCustomEditContolAction** action class.

This action class is created in the ActionList and its property AttachedControl is set to the TWPComboBox instance you need to attach. The property AttachedControlStyle to select the functionality is not used for TWPComboBox - they have their own property ComboboxStyle for the same purpose.

Create shortcuts, such as *Ctrl+I* to activate/deactivate ITALIC:

To do this You can use the event OnKeyPress.

```
procedure TForm1.WPRichText1KeyPress(Sender: TObject;
  var Key: Char);
  procedure ToggleStyle(sty : TOneWrtStyle);
  begin
     if sty in WPRichText1.CurrAttr.Style then
          WPRichText1.CurrAttr.DeleteStyle([sty])
     else WPRichText1.CurrAttr.AddStyle([sty]);
     WPRichText1.SetFocusValues(true);
  end;
begin
  if Key=Char(Integer('B')-64) then // Ctrl + B
  begin
     ToggleStyle(afsBold);
     Key := #0;
  end else
  if Key=Char(Integer('I')-64) then // Ctrl + I
  begin
     ToggleStyle(afsItalic);
     Key := #0;
  end else
  if Key=Char(Integer('U')-64) then // Ctrl + U
  begin
     ToggleStyle(afsUnderline);
     Key := #0;
  end;
end;
```

Note

Please do not use actions if you want to execute certain methods of the TWPRichText component from your own code. You can call the procedure directly (i.e. WPRichText1.Save) or change text attributes using CurrAttr (i.e. WPRichText1.CurrAttr.AddStyle[afsBold]).

5.3.5 Localization

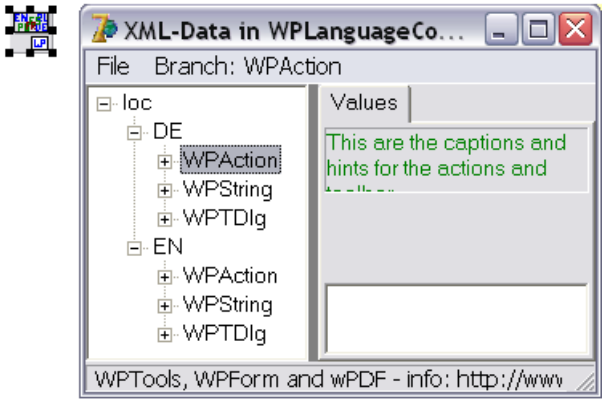
For anyone wanting to work with inch units, please add the following line in your code (for example, in the FormCreate event).

```
GlobalValueUnit := euInch;
```

WPTools 5 supports the localization of the texts which are used for filters and error messages and also the localization of the provided property dialogs.

The localized strings have to be stored in the **TWPLanguageControl** which is part of the WPShared package. To **install** the WPShared package please add the unit wpshared.pas to the WPTools 5 package.

Please drop this component and your main form and after a double click, load/merge the XML source files which contain the translated strings:



In WPTools 5 you need to add a few lines of code to activate the localisation since the WPShared and WPTools packages had to be kept strictly seperate. The properties of the TWPLanguageControl, "Active" and "AutoLoadStrings" do not work.

To active use this code in the OnCreate event of the form:

```
// requires unit WPUtils and WPActnStr
WPLangInterface := TWPLocalizationInterface.Create(WPLanguageControl1);
WPLanguageControl1.GlobalLanguage := 'DE';
WPLocalizeLoadForms := TRUE;
WPTools_LoadVCLStrings;
WPTools_LoadActionStrings;
```

In the OnDestroy event add this:

```
WPLangInterface.Free;
```

To change the language at runtime use code like this:

```
WPLanguageControl1.GlobalLanguage := 'DE';
WPLocalizeLoadForms := TRUE;
WPTools_LoadVCLStrings;     // from unit WPUtil
WPTools_LoadActionStrings; // from unit WPActnStr
```

How does the localization work?

In WPTools 5 we are using a localization interface which is defined as:

```
IWPLocalizationInterface = interface
    ['{A12EF1F7-E592-4483-855F-67E28332AFC5}']
// This method can be used to save the menu items and captions
   on a certain form. If you use the TWPLocalizeForm class you don't need
   to care about that. }
    procedure SaveForm(
      const Name: string;
      Form: TWinControl;
      Menus, Captions, Hints: Boolean);
// Load all Components on a certain TForm. }
    procedure LoadForm(
      const Name: string;
      Form: TWinControl;
      Menus, Captions, Hints: Boolean);
// This method saves a string list under a certain name. The string list has to use
   the syntax NAME=xxx\n }
    procedure SaveStrings(
      const Name: string;
      Entries: TStrings;
      Charset: Integer);
// Loads back the string(s) saved with  WPLangSaveStrings }
    function LoadStrings(
      const Name: string;
      Entries: TStrings;
      var Charset: Integer): Boolean;
// Method to save a certain string. To save multiple strings use  WPLangSaveStrings
    procedure SaveString(
      const Name, Text: string;
      Charset: Integer);
// Loads back a string saved with WPLangSaveString
    function LoadString(
      const Name: string;
      var Text: string;
      var Charset: Integer): Boolean;
  end;
```

This interface is implemented by the TWPLanguageControl. The TWPLocalizeForm (implemented in unit WPUtil, it is the anchestor of all localizable dialogs) automatically uses this interface through the instance of the TWPLocalizationInterface class which must be created by your code:

```
WPLangInterface := TWPLocalizationInterface.Create(WPLanguageControl1);
```

5.3.6 Layoutmodes

The different layout modes are probably one of the most exciting feature in WPTools 5. You will hardly find a component which offers this kind of versatility in a text editing tool.

Different layout modes are mainly activated in property **LayoutMode**
- but also the properties PageColumns, AutoZoom, Zooming, ViewOptions and OnPageGapGetText are important for the display of the text.

If you are using our multiview technology (multiple editors show one text), each editor can use different settings for the mentioned properties. So it is possible that one editor displays the thumbnails while a different editor shows the text in 'normal' mode!

Changing the layout modes is also extremely fast, usually the already formatted pages are rearranged on the virtual desktop!

Please note that the TWPPreview component also inherits from the usual editor component. So it has the same properties. But it also introduces the property SinglePageMode. If this property is true, unlike the default display of all pages in a row, only one row of pages will be displayed. BTW - if you are using the TWPPreview or TWPPreviewDlg you are already using the multiview technology.

Possible values of the property *LayoutModes* are:

wplayNormal : Only show the text area without margins - do not paint background color.

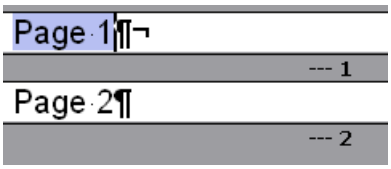
wpWordWrapView : Display like wplayNormal - used when the word wrap is set to the editor box.

wplayShowManualPageBreaks - like wpNormal but draws a dashed line for page breaks

wplayPageGap - display the text similar to wplayNormal with a bar iin between

wplayExtendedPageGap - works like wplayPageGap but does not suppress the left and right margin.

wplayPageGap example:



The display of the page numbers ("--- 1") is activated using the ViewOption wpShowPageNRinGap. You can use the event OnGetPageGapText to display a different text.

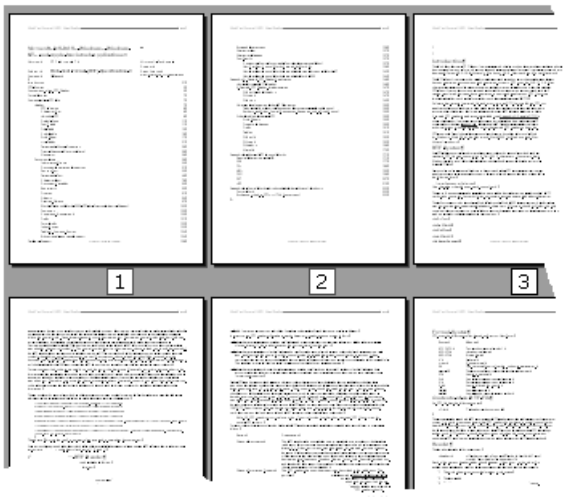
wplayShrinkedLayout - show PageLayout without header and footer (reduces page height!)

wplayLayout - show PageLayout but do not paint header and footer.

wplayFullLayout - show PageLayout with header and footer.

wpDualPageView - display 2 pages side by side. A similar effect can be accieved by setting the property "PageColumns" to 2 but wpDualPageView makes sure the 2 pages are handled as 1 by the auto zoom function.
The DualPage view expects the first page to be ouside and the 2nd and 3rd to be side by side. If the property WPRichText.Memo.DualPageViewAlternate is true the first page is displayed side by side with the second.

wpThumbNailView - display thumb nails of the pages, optionally with display of the page numbers in little boxes. This is activated by *ViewOption* wpShowPageNRinGap.



property ViewOptions

The following values are possible:

wpShowGridlines - draw a gray line for table borders which would mbe otherwise invisible

wpDisableHotStyles - disable the hot styles (or hover styles) which can be activated for hyperlinks or fields.

wpShowCR - show a ¶ symbol at the end of a paragraph

wpShowFF - displays ¶¬ at the end of a paragraph when the next paragraph starts on a new page

wpShowNL - displays an arrow for a new line

wpShowSPC - shows a dot for the code #32 (SPACE)

wpShowHardSPC - shows a dot for the code #160 (non breaking space)

wpShowTAB - show an arrow in the place of tabstops (suppressed if fillsigs are active)

wpShowParCalcNames - Display the names assigned using property WPAT_PAR_NAME

wpShowParCalcCommands - Display the formulas assigned to paragraphs and cells.

wpShowParNames- Display the names assigned using property TParagraph.Name

wpNoEndOfDocumentLine - display a line at the end of the document if not in pagegap mode. Ignore the typo 'No'.

wpHideSelection - Always hides the selection

wpHideSelectionNonFocussed - hides thenselection when editor does not have the focus

wpShowPageNRinGap - displays a number or any other text provided by event OnPageGapGetText either at the right border or in a box under the page

wpDrawFineUnderlines - always draw thin underlines

wpDontGrayHeaderFooterInLayout - do not shaed the header and footer texts

wpInfiniteTextArea - makes it possible to show a text as if it is infinite. This can be used for a scroller control, for example to show news or credits. To scroll change the property TopOffset in a timer event.

wpDontPaintPageFrame - with page layout modes, do not draw a frame around the page

wpCenterPaintPages - center the page horizontally in the window. This is useful for preview windows.

wpUseOwnDoubleBuffer - Usually a shared double buffer is used for all editors to limit the memory use - unless thumbnailmode has been activated or this flag is active.

property AutoZoom

The following values are possible:

wpAutoZoomOff - use the value of property Zooming to change the aspect ratio of the text display

wpAutoZoomWidth - automatically adjust the aspect ratio to make room for the complete width of the page

wpAutoZoomFullPage - automatically adjust the aspect ratio to make room for the complete size of the page

wpAutoZoomAdjustColumnCount - automatically adjust the property PageColumns to show as many pages side by side as fit into the window. Usually the property Zooming should be set to a small value, for example 30.

wpAutoZoomAsManyAsPossibleInRow - show as many pages side by side but allow a different count of pages each row. You can see the effect in the lower editor in the section demo. This mode should be also combined with a small zooming value.

Please note that the property CurrentZooming can be read to get the current aspect ratio as a floating point multiplicator.

5.3.7 Header and Footer

WPTools provides powerful support for headers and footers.
You can create a header or footer which is valid for all pages or header and footers for the first, the odd and the even pages.

In WPTools 5 ist is also possible to start a new section in the text. This section can then have its own set of header and footer and also, optionally, a different page size or different margins.

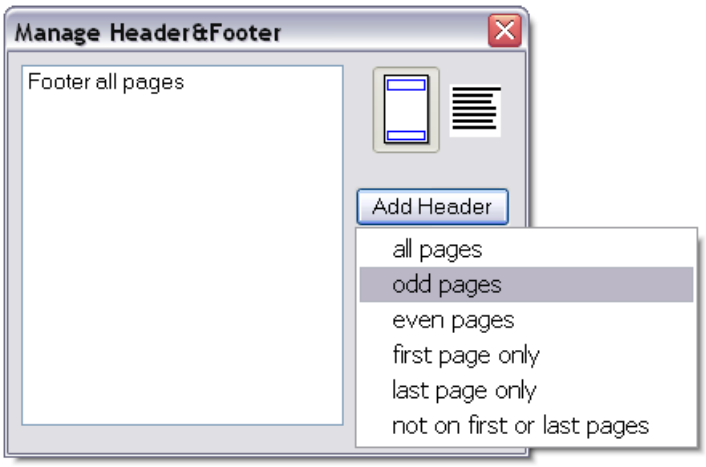
Example: Create a Footer with the text "PAGE # of ##"

```
var par : TParagraph;
    txtplain : Cardinal;
begin
  par:=WPRichText1.HeaderFooter.Get(wpFooter,wpraOnAllPages).FirstPar;
  WPRichText1.WritingAttr.Clear;
  WPRichText1.WritingAttr.SetFontName('Courier New');
  WPRichText1.WritingAttr.SetFontSize(900);  // 9pt
  txtplain := WPRichText1.WritingAttr.CharAttr;
  par.ClearText;
  par.ASet(WPAT_Alignment, Integer(paralRight));
  par.Insert(0,'Page ',txtplain);
  par.InsertNewObject(maxint, wpobjTextObject, false, false,txtplain).Name :=
WPTextFieldNames[wpoPageNumber];
  par.Insert(maxint,' of ',txtplain);
  par.InsertNewObject(maxint, wpobjTextObject, false, false,txtplain).Name :=
WPTextFieldNames[wpoNumPages];
end;
```

Please note that the property 'FirstPar' will always create a paragraph if the text block was empty.

"Manage Header and Footer", a useful dialog

WPTools comes with a useful form in unit WPManHeadFoot to work with header and footer.



This form makes it easy to navigate to a certain header or footer. It can be edited, copied or deleted. The two buttons switch between the page layout mode and the normal layout mode. If the user clicks on a header or footer which is currently not visible, the text cannot be edited in page layout mode, so the dialog will automatically select the normal mode.

To use this form add the unit to the uses clause and create a global variable:

```
ManHeadFoot : TWPManageHeaderFooter;
```

Now, to show the form in a non-modal:

```
if ManHeadFoot=nil then
   ManHeadFoot := TWPManageHeaderFooter.Create(Self);
ManHeadFoot.WPRichText := WPRichText1;
ManHeadFoot.Show;
```

We recommend to check out the source of this dialog since it also contains good example code.

Properties T*WPRichText*.WorkOnText and *TWPRTFDataBlock*.WorkOnText

This property selects the text which is currently **edited**. All commands will operate on the selected text part. Only ‚Load' and ‚Save' will automatically select the body.

Possible values for 'WorkOnText' are

```
wpIsBody
wpIsHeader
wpIsFooter
```

If you need to select the special text for a certain 'range' you the property HeaderFooterTextRange. (type TWPPagePropertyRange).

```
WPRichText1.HeaderFooterTextRange := wpraOnOddPages;
WPRichText1.WokOnText := wpIsHeader;
```

Now all the procedures work with the header for odd pages. You can execute mailmerge "MergeText", add number fields "InputTextFieldName('PAGE')" or inserts texts "InputString'Hello World')".

If you have a reference to a certain **TWPRTFDataBlock** - for example provided by WPRichText1.HeaderFooter.Get - you can also set the boolean property WorkOnText of this element to TRUE to select it for editing!

```
WPRichText1.HeaderFooter.Get(wpIsHeader,wpraOnFirstPage, '').WorkOnText := TRUE;
WPRichText1.InputString( 'WPTools Documentation');
```

```
WPRichText1.WorkOnText := wpIsBody;
```

Please note that you will only see the selected text exclusively in the 'normal' LayoutMode. Otherwise the cursor simply moves to the header or footer region.

Notes
a) Please note, the above puts the cursor into a certain header/footer text. It does not change which header or footer is currently displayed on the page. If you need to modify the way header and footer are selected for the display in page layout mode use the **OnGetSpecialText** event.

This event handler selects the ALL PAGES header or footer for all pages, also first, odd and even no matter what other header or footer are there:

```
procedure TForm1.WPRichText1GetSpecialText(Sender: TObject;
  par: TParagraph; PosInPar, PageNr: Integer; Kind: TWPPagePropertyKind;
  var IsLastPage, UseThis: Boolean; var SpecialText: TWPRTFDataBlock);
begin
  // We use 'Get' to find the header or footer
  SpecialText := WPRichtext1.HeaderFooter.Get(Kind, wpraOnAllPages, '');
  // Only then the variable is actually used
  UseThis := TRUE;
end;
```

b) It is important to note that if the PrintHeaderFooter sub-property is set to wprNever, the headers and footers will not be displayed on screen either.
Also, you can choose if the headers and footers will be displayed in grayed form by setting the wpDontGrayHeaderFooter sub-property of the ViewOptions property to false.

c) The header and footer can be printed in the top/bottom margin area or in the text area. It is up to you what you like better, both possibilities are used by standard word processors. This behavior is controlled by the sub-property PrintHeaderFooterInPageMargins in property HeaderHeader. In the property Header you can also modify the values for the margins and page and the default values that should be set when the buffer was cleared.

Collection HeaderFooter : TWPRTFDataCollection

The headers and footers (and also the body and optional texts) are stored in the runtime collection property TWPRichText.HeaderFooter.

This collection manages a list of instances of the class TWPRTFDataBlock. Each instance holds one 'special' text. This text can be used as a header or a footer within the selected range.

```
  TWPRTFDataBlock = class(TCollectionItem)
  ...
  published
    property Name: string
    property UsedForSectionID
    property Range: TWPPagePropertyRange
    property Kind: TWPPagePropertyKind
    property RtfText: TWPRTFDataContents
  end;

  TWPPagePropertyRange =
    (wpraOnAllPages, wpraOnOddPages, wpraOnEvenPages, wpraOnFirstPage,
    // these Modes are not compatible to the RTF Standard !
    // They have priority !!!!
    wpraOnLastPage, wpraNotOnFirstAndLastPages,
    wpraNamed, wpraIgnored);

  TWPPagePropertyKind = (wpIsBody, wpIsHeader, wpIsFooter,
      wpIsFootnote,         // The last 4 are for internal use only
      wpIsLoadedBody,
      wpIsDeleted,
      wpIsOwnerSelected);
```

```
TWPRTFDataContents = class(TPersistent)
  public
    constructor Create(Source: TWPRTFDataBlock);
    procedure LoadFromStream(Stream: TStream); virtual;
    procedure SaveToStream(Stream: TStream); virtual;
    property  AsString: string read GetAsString write SetAsString;
    procedure Assign(Source: TPersistent); override;
    property  Format: string read FFormat write FFormat;
  end;
```

To change the text you can load the text from a stream using Item.RtfText.LoadFromStream or you can simply set or read the property AsString! This string can also be in HTML or WPTOOLS format!

The RTFDataCollection provides this functions to find certain header or footer entries. You can of course enumerate all the items using the Items[] array.

```
function Find(Kind: TWPPagePropertyKind; Range: TWPPagePropertyRange;

const Name: string = '*'; UsedForSectionID: Integer = 0): TWPRTFDataBlock;

procedure DeleteTexts(UsedForSectionID: Integer); overload;

procedure DeleteTexts(Kind: TWPPagePropertyKind; UsedForSectionID: Integer);
overload;

procedure DeleteRTFData(RTFData: TWPRTFDataBlock);

procedure DeleteText(Kind: TWPPagePropertyKind; Range: TWPPagePropertyRange;
UsedForSectionID: Integer = 0);
```

We suggest to use this code to add a new header:

```
WPRichText1.HeaderFooter.Get(
    wpIsHeader,
    wpraOnAllPages,
    '').RtfText.AsString
     :=  WPEditor.AsString; // any other TWPRichText which is used to edit the
header
```

-or-

```
  WPRichText1.HeaderFooter.Get(
       wpIsHeader,
       wpraOnAllPages, '').RtfText.LoadFromStream( filestream1 );
```

C++Builder:

```
WPRichText1->HeaderFooter->Get(wpIsHeader,wpraOnAllPages, "")->RtfText->AsString;
```

Event: OnGetSpecialText

This event receives this parameters:

```
Sender: TObject;
par: TParagraph;
PosInPar: Integer;
PageNr: Integer;
Kind: TWPPagePropertyKind;
var IsLastPage: Boolean;
var UseThis: Boolean;
var SpecialText: TWPRTFDataBlock
```

You can set UseThis to true to let the RTF Engine use as header or footer whatever item you have choosen in 'SpecialText'. To switch off the header or footer set this parameter to nil:

```
SpecialText := nil;
if PageNr=1 then
```

```
UseThis := TRUE
  else UseThis := FALSE;
```

5.3.8 Create Sections

Sections make it possible to use many different header and footer texts and different page sizes in one document.

The editor will display an arrow  in the left margin where a new section starts.

The following code can be used to create a new section:

```
var sectionprops : TWPRTFSectionProps;
begin
  // New Page
  WPRichText1.InputString(#12);
  // New section properties
  sectionprops := WPRichText1.ActiveParagraph.StartNewSection;
  // Now we can do something with sectionprops
  ...
end;
```

Each section is defined by an instance of TWPRTFSectionProps. The property WPRichText.Header which stores the default page size for the whole document is consequently implemented using a class which inherits from TWPRTFSectionProps.

Usually all instances of TWPRTFSectionProps use the property value defined in the master section property (WPRichText.Header). If certain attribute should be unique for a section the property Selected must be set accordingly.

```
  sectionprops.Select := [wpsec_PageSize];
  sectionprops.Landscape := TRUE;
```

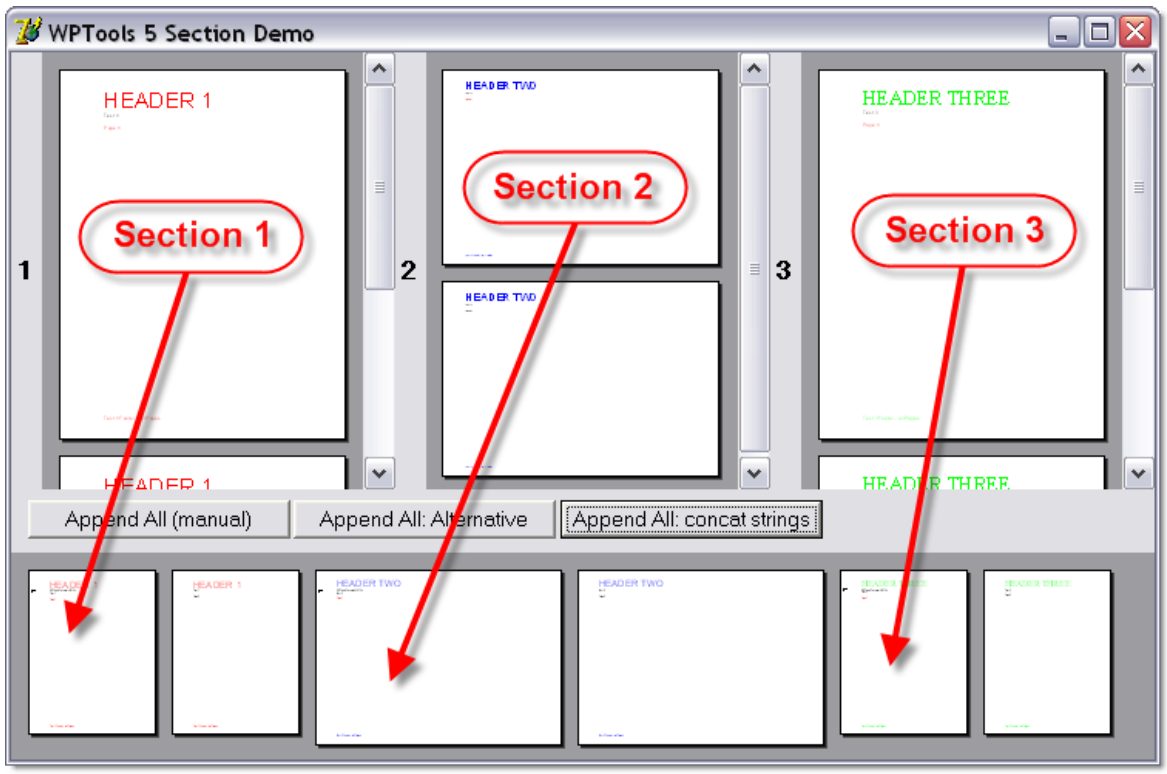
The following flags exist in property Select:

wpsec_PageSize	--> Overwrite PageWidth, PageHeight and Landscape
wpsec_Margins	--> Top, left and other margins
wpsec_TabDefault	--> Change deftabstop property
wpsec_PageMirror	--> change the marginmirror property

Note: The "speed reformat" feature of WPTools 5 does not work well with different page sizes yet. So please switch this feature off using WPAll.FormatOptions := [wpDisableSpeedReformat];

It is also possible to select that certain header or footer texts should be only used for one section. This is done by the property TWPRTFDDataBlock.UsedForSectionID. If this property is changed to sectionprops.SectionID the text block will be only used for that special section.

The demo AppendAsSection shows how texts from different editors (WP1,WP2, WP3) can be appended to create one multi section document in the editor WPALL:



The 3 buttons use different approaches:

1.) Full implementation - shows how to work with sections

DELPHI CODE

```
procedure TWPALL.AppendClick(Sender: TObject);
   procedure AppendText(Source : TWPRTFDataCollection);
   var sectionprops : TWPRTFSectionProps ;
       i    : Integer;
       textblock : TWPRTFDataBlock;
   begin
     // Create a new Page if required
     if WPAll.IsEmpty then
          WPAll.CheckHasBody
     else WPAll.InputString(#12);
     // Create new section properties
     sectionprops := WPALL.HeaderFooter.AddSectionProps;
     // Assign the default page size
     sectionprops.Assign(Source.Header);
     sectionprops.Select := [wpsec_PageSize,wpsec_Margins];
     // Copy all header + footer into certain section
     for i:=0 to Source.Count-1 do
       if Source[i].Kind in [wpHeader, wpFooter] then
       begin
         textblock := WPALL.HeaderFooter.Append(
           Source[i].Kind,
           Source[i].Range,
           Source[i].Name);
         textblock.UsedForSectionID := sectionprops.SectionID;
         textblock.RtfText.Assign(Source[i].RTFText);
       end;
     // The current paragraph starts this section
```

```
WPAll.ActiveParagraph.SectionID := sectionprops.SectionID;
     include(WPAll.ActiveParagraph.prop,paprNewSection);
     // Copy the text as part of a certain section
     WPAll.CPPosition := MaxInt;
     WPAll.SelectionAsString := Source.AsANSIString('WPTOOLS');
   end;
begin
  WPAll.Clear;
  AppendText(WP1.HeaderFooter);
  AppendText(WP2.HeaderFooter);
  AppendText(WP3.HeaderFooter);
end;
```

C++BUILDER Example:

This code appends the text from a different editor to "WPRichText1" as a new section

```
void __fastcall TWPALL::AppendNewSection(TWPRTFDataCollection * Source)
{
 TWPRTFSectionProps * sectionprops;
 TWPRTFDataBlock * textblock ;
 int i;

 // Create a new Page if required
 if (this->WPRichText1->IsEmpty())
 this->WPRichText1->CheckHasBody();
 else this->WPRichText1->InputString( "\f",0);

 // Create new section properties
 sectionprops = this->WPRichText1->HeaderFooter->AddSectionProps();

 // Assign the default page size
 sectionprops->Assign(Source->Header);

 //sectionprops->Select = [wpsec_PageSize,wpsec_Margins];
 sectionprops->Select << wpsec_PageSize << wpsec_Margins ;

 // Copy all header + footer into certain section
 for (i=0; i < Source->Count; i++ )
{
 TWPRTFDataBlock * src = Source->Items[i] ;
 if ( src->Kind == wpHeader || src->Kind == wpFooter )
        {
        textblock = this->WPRichText1->HeaderFooter->Append(
        Source->Items[i]->Kind ,
        Source->Items[i]->Range ,
        Source->Items[i]->Name );
        textblock->UsedForSectionID = sectionprops->SectionID;
        textblock->RtfText->Assign(Source->Items[i]->RtfText);
        }
 }//for()

 // The current paragraph starts this section
 this->WPRichText1->ActiveParagraph->SectionID = sectionprops->SectionID;

 //include(WPAll->ActiveParagraph->prop,paprNewSection);
 this->WPRichText1->ActiveParagraph->prop << paprNewSection ;

 // Copy the text as part of a certain section
 this->WPRichText1->CPPosition = MaxInt;
 this->WPRichText1->SelectionAsString = Source->AsANSIString("WPTOOLS");
}
```

2.) Use utility procedure AppendAsSection - same functionality as 1)

```
procedure TWPALL.Append2Click(Sender: TObject);
begin
  WPAll.HeaderFooter.AppendAsSection(WP1.HeaderFooter);
  WPAll.HeaderFooter.AppendAsSection(WP2.HeaderFooter);
  WPAll.HeaderFooter.AppendAsSection(WP3.HeaderFooter);
end;
```

3) Use strings in WPTOOLS format with the <newsection/> tag.
This technique allows it to create a multi section text by simply appending strings or streams without loading the text into an editor. This shows how versatile the WPTOOLS format can be used:

```
procedure TWPALL.AppendWithStringsClick(Sender: TObject);
begin
  WPAll.AsString := '<newsection/>' + WP1.AsANSIString('WPTOOLS')
   +'<newsection/>' + WP2.AsANSIString('WPTOOLS')
   +'<newsection/>' + WP3.AsANSIString('WPTOOLS');
  // Using <newsection pagebreak=0/> no page break will be inserted
end;
```

5.3.9 Use WPTools5 with TBX (Toolbar2000 Extension)

Toolbar2000 is a component by Jordan Russel to build a modern GUI.
It can be extended with TBX, written by Alex A. Denisov.
You can download and order ToolBar2000 and TBX from
www.g32.org and www.jrsoftware.org. For additional TBX themes check out www.rmklever.com.

It is easy to create an impressive word processing application using this 2 products and of course, WPTools 5.

INFO: *Using the WPTools 5 Demo VCL You cannot activate the support for TBX combos using the define USETBX. Therefore the demo does not install the TBX demo project.*

1) Please activate the define USETBX in file WPINC.INC or, better, in your project options.

This modifies the compilation of the units wpruler and wpaction. Please make sure you do a 'compile all' after the change.

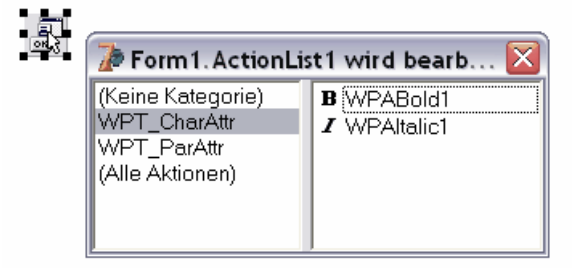
2) Create a toolbar (see Toolbar2000 and TBX manual)

Example:

3) Now you can bring life to the tool items

a) For the buttons, such as 'bold'

Create a TActionlist on the form. Assign it to the property 'Actionlist' of the editor component, TWPRichText.
In this action list you can create WPTools standard actions, such as TWPABold



This new actions can be assigned to the 'Action' property of menu items and button elements. Now

they will automatically work with the attached TWPRichText.

b) To activate the color drop downs more code is requires.

If you use 3 drop downs for text color, highlight color and paragraph background color use the tags 1, 2 and 3.
Depending on the tags the drawing and OnChange code can be behave differently.

This code is used in the OnDrawImage event of the button itself.

```
procedure TForm1.bColorButtonDrawImage(Item: TTBCustomItem;
  Viewer: TTBItemViewer; Canvas: TCanvas; ImageRect: TRect;
  ImageOffset: TPoint; StateFlags: Integer);
var
  DC: HDC;
  Color: TColor;
  ColorInd: Integer;
begin
  DC := Canvas.Handle;
  ColorInd := -1;
  if not Boolean(StateFlags and ISF_DISABLED) then
  begin
    case Item.Tag of
      1: ColorInd := WPRichText1.CurrAttr.GetColorEx(WPAT_CharColor);
      2: ColorInd := WPRichText1.CurrAttr.GetColorEx(WPAT_CharBGColor);
      3: ColorInd := WPRichText1.CurrAttr.GetColorEx(WPAT_FGColor);
    else ColorInd := -1;
    end;
    OffsetRect(ImageRect, ImageOffset.X, ImageOffset.Y);
    ImageRect.Top := ImageRect.Bottom - 4;
    if ColorInd >= 0 then
    begin
      Color := WPRichText1.CurrAttr.NrToColor(ColorInd);
      Canvas.Brush.Color := Color;
      Canvas.FillRect(ImageRect);
    end
    else
    begin
      FrameRectEx(DC, ImageRect, clBtnShadow, True);
      DitherRect(DC, ImageRect, clBtnFace, clBtnShadow);
    end;
  end;
end;
```

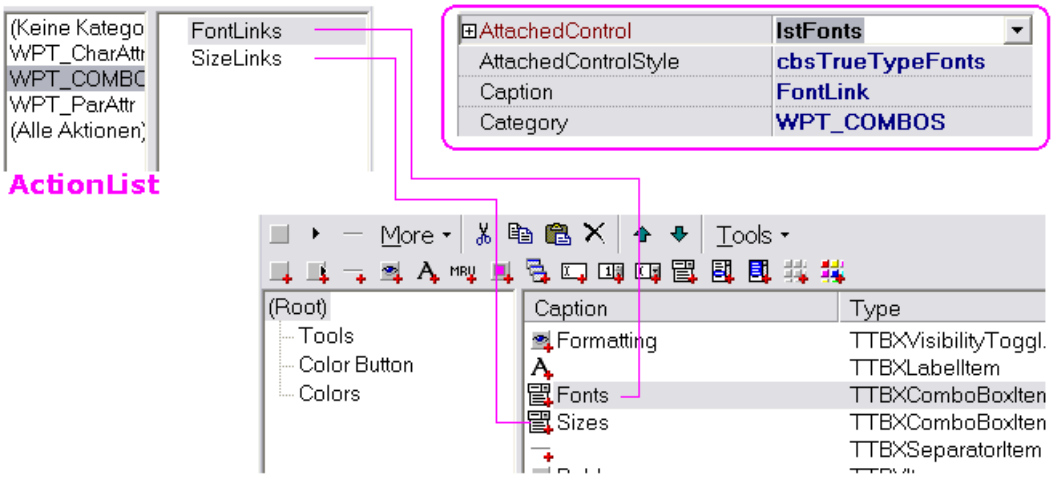
Each of the buttons has as subitems (which are used for the dropdown) and ColorPalette.
We assigned the tags 1,2 and 3 to this 3 color palettes as well and use the same OnChange code.

```
procedure TForm1.TBXColorPalette1Change(Sender: TObject);
var aColor: Integer;
begin
  aColor := (Sender as TTBXColorPalette).Color;
  case (Sender as TTBXColorPalette).Tag of
    1: WPRichText1.CurrAttr.Color :=
      WPRichText1.CurrAttr.ColorToNr(aColor, true);
    2: WPRichText1.CurrAttr.BKColor :=
      WPRichText1.CurrAttr.ColorToNr(aColor, true);
    3: WPRichText1.CurrAttr.ParColor :=
      WPRichText1.CurrAttr.ColorToNr(aColor, true);
  end;
end;
```

This was already sufficient to make all three color buttons work.

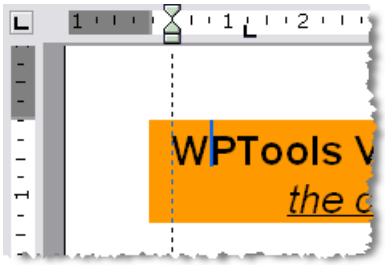
c) to bring life to the drop down combos, such as the font selection, you only need to use the special WPTools' pseudo actions (TWPToolsCustomEditContolAction) which have to be added to the action list as well.

After such an action has been created in category 'WPT_COMBOS', you can select the style in property 'AttachedControlStyle' and the list box in property 'AttachedControl'.
The first time the drop down will be used it will be initialized and the events to modify the attached editor will be set. Please note that the combo box styles cbsColor, cbsBKColor, cbsParColor are not supported by TWPToolsCustomEditContolAction. The style cbsStandard is the neutral value.



Note: The TWPToolsCustomEditContolAction can be also used to attach TWPComboBox elements. In this case 'AttachedControlStyle' has to be set to cbsStandard.

Now you can create the horizontal and if required, also the vertical ruler:



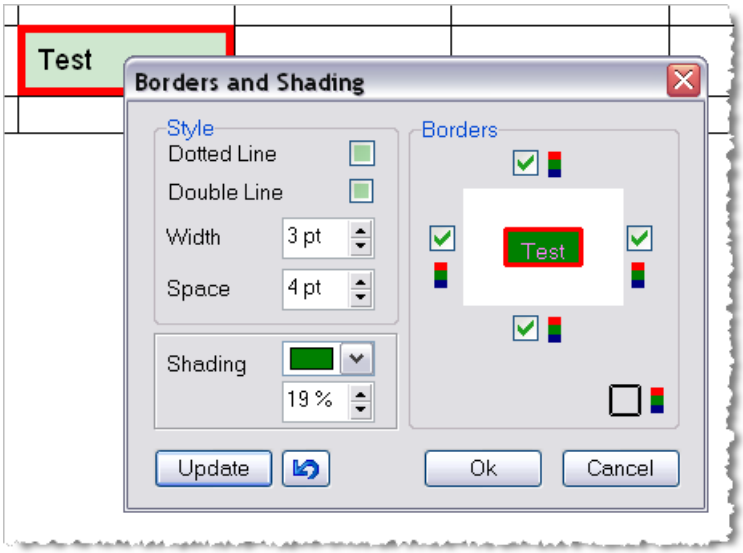
You only need to attach the ruler controls to the the property **WPRuler** and **VertRuler** of the TWPRichText Control.
If you are using a vertical ruler disable the flag wpNoVertRulerAttached in WPRuler.Options.

How to use the predefined dialogs?

Please drop one of the dialog components on the form:

Then set the property 'EditBox' to point to the editor which should be used.

Using a call to the method 'Execute' the dialog can be opend:



(Screenshot of the Border dialog. When this dialog is opend all properties are set to 'undefined': the user can change any of the elements and leave the other unchanged)

5.3.10 Set Attributes in code (unit WPCreateDemoText)

WPTools can be perfectly used to create text and tables in code. This text can be saved in RTF or WPTOOLS format, printed and (with wPDF) exported to PDF.

In unit WPCreateDemoText you will find a procedure which creates text with different character and paragraph attributes. We have included this procedure in this chapter and inserted screenshots of the output which is created by each part of the code.

The procedure starts with the variable declaration. Most important is par: TParagraph which is used as reference to the current paragraph while the text is created. Each new paragraph (or table) is created after this paragraph and the reference is changed to this new object.

cha: TWPStoredCharAttrInterface is used to create character attribute index values which are then used in par.SetText and par.Append methods.

```
procedure CreateDemoText(RTFMemo : TWPCustomRtfEdit; Mode : TWPCreateDemoText);
var par: TParagraph; // The current paragraph or table
  row, cell: TParagraph; // the current rows and cells
  cha: TWPStoredCharAttrInterface;
  HeadlineA, TextA, RedTextA: Cardinal;
```


```
  procedure CreateEmptyPar;
  begin
    par := par.AppendNewPar(true);
    par.LoadedCharAttr := TextA; // Default attribute for empty paragraphs
  end;
begin
```

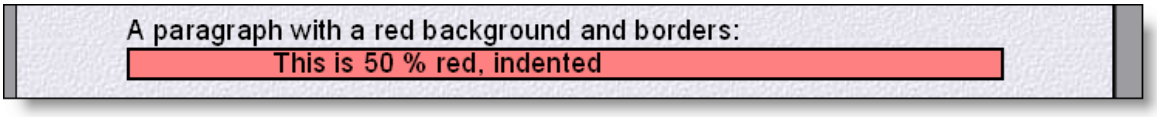
Initialize character attributes for the text

```
cha := RTFMemo.AttrHelper;
cha.Clear;
cha.SetFontName('Times New Roman');
cha.SetFontSize(12);
HeadlineA := cha.CharAttr;

cha.Clear;
cha.SetFontName('Arial');
cha.SetFontSize(11);
TextA := cha.CharAttr;

cha.SetColor(clRed);
RedTextA := cha.CharAttr;
```

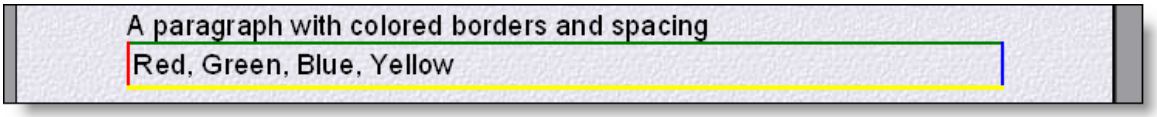
Initialize the document and create the paragraph "par"

```
if Mode=wpNewText then // Clear and set page size and margins
begin
RTFMemo.Clear;

RTFMemo.Header.SetPageWH(
  WPCentimeterToTwips(15),
  WPCentimeterToTwips(29.7),
  WPCentimeterToTwips(1.5),
  WPCentimeterToTwips(1.5),
  WPCentimeterToTwips(1.5),
  WPCentimeterToTwips(1.5)
  );
RTFMemo.CheckHasBody;
par := RTFMemo.FirstPar;
end
else if Mode=wpInsertText then  // insert at current position
begin
  par := RTFMemo.InsertPar;
end
else if Mode=wpAppendText then // append at end of text
begin
  par := RTFMemo.ActiveText.LastPar;
  CreateEmptyPar;
end;
```

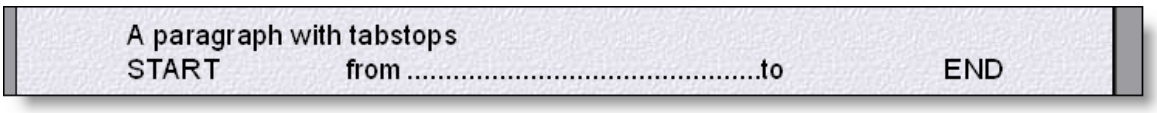

```
// Set the text of the first paragraph
par.SetText('A paragraph with a green background:', HeadlineA);

// append a new, clean paragraph and set attributes
par := par.AppendNewPar(true);
par.ASetColor(WPAT_BGColor, clGreen);
par.ASet(WPAT_ShadingValue, 30);
par.SetText('This is 30 % green', TextA);

CreateEmptyPar;
```




```
  par := par.AppendNewPar(true);
  par.SetText('A paragraph with a red background and borders:', HeadlineA);
  // append a new, clean paragraph and set attributes
  par := par.AppendNewPar(true);
  // Indents
  par.ASet(WPAT_IndentLeft, WPCentimeterToTwips(2));
  par.ASet(WPAT_IndentRight, WPCentimeterToTwips(2));
  // Border
  par.ASet(WPAT_BorderFlags,
    WPBRD_DRAW_Left + WPBRD_DRAW_Right + WPBRD_DRAW_Top + WPBRD_DRAW_Bottom); // =
WPBRD_DRAW_All4
  par.ASet(WPAT_BorderWidth, WPCentimeterToTwips(0.05)); // 0,5 mm

  par.ASetColor(WPAT_BGColor, clRed);
  par.ASet(WPAT_ShadingValue, 50);
  par.SetText('This is 50 % red, indented', TextA);
  CreateEmptyPar;
```




```
  par := par.AppendNewPar(true);
  par.SetText('A paragraph with colored borders and spacing', HeadlineA);
  par := par.AppendNewPar(true);
  par.ASet(WPAT_SpaceBefore, WPCentimeterToTwips(0.1)); // 1 mm  padding
  par.ASet(WPAT_SpaceAfter, WPCentimeterToTwips(0.1)); // 1 mm  padding
  par.ASet(WPAT_IndentLeft, WPCentimeterToTwips(0.1)); // 1 mm  padding
  par.ASet(WPAT_SpaceBefore, WPCentimeterToTwips(0.1)); // 1 mm  padding

  par.ASet(WPAT_BorderFlags, WPBRD_DRAW_All4); // all lines
  par.ASet(WPAT_BorderWidth, WPCentimeterToTwips(0.05)); // 0,5 mm
  par.ASetColor(WPAT_BorderColorL, clRed); // Left
  par.ASetColor(WPAT_BorderColorT, clGreen); // Top
  par.ASetColor(WPAT_BorderColorR, clBlue); // Right
  par.ASetColor(WPAT_BorderColorB, clYellow); // Bottom
  par.SetText('Red, Green, Blue, Yellow', TextA);
  CreateEmptyPar;
```



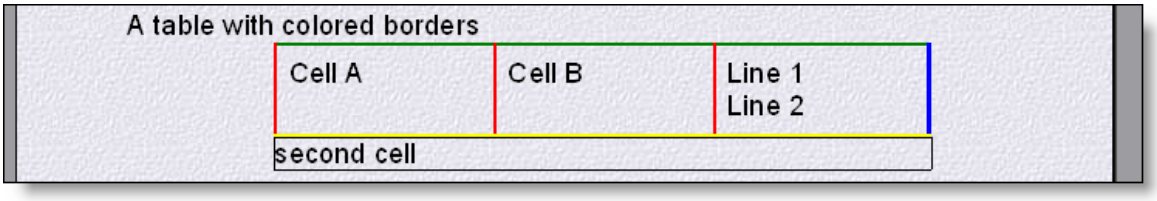

```
  par := par.AppendNewPar(true);
  par.SetText('A paragraph with tabstops', HeadlineA);
  par := par.AppendNewPar(true);
  par.TabstopAdd(WPCentimeterToTwips(3), tkLeft);
  par.TabstopAdd(WPCentimeterToTwips(9), tkRight, tkDots);
  par.TabstopAdd(WPCentimeterToTwips(12), tkRight, tkNoFill);

  par.Append('START', RedTextA);
  par.Append(#9 + 'from' + #9 + 'to' + #9, TextA);
  par.Append('END', RedTextA);
  CreateEmptyPar;
```

```
  // CREATE A HORIZONTAL LINE
```

```
CreateEmptyPar;
  with par.AppendNewObject(wpobjHorizontalLine) do
  begin
    IParam := clGreen;
    // Width  := WPCentimeterToTwips(10);
    CParam := -WPCentimeterToTwips(1);
  end;
  CreateEmptyPar;
```

horizontal lines can use the following parameters:
Width: width in twips, if 0 the printable area is used
CParam:the offset to the margins of the printable area. Native values are possible
IParam: the color as RGB value
Height: the height in twips

Note: AppendNewObject() and AppendNewObjectPair() can also be used to create hyperlinks.



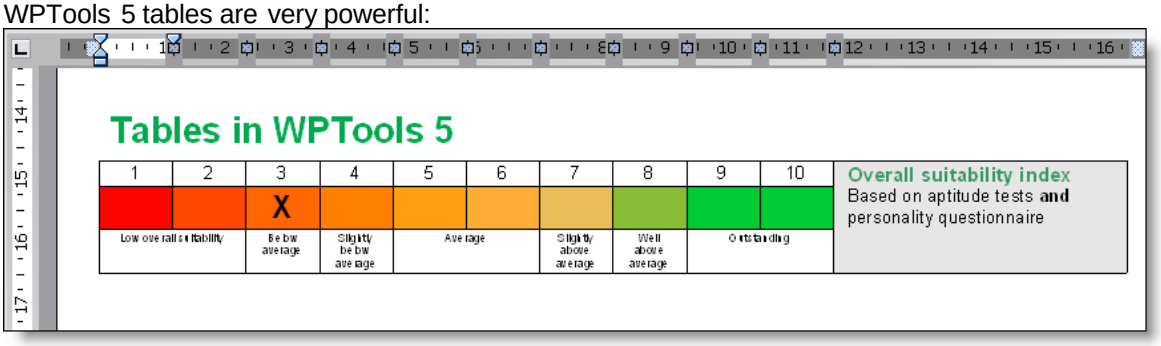

```
  par := par.AppendNewPar(true);
  par.SetText('A table with colored borders', HeadlineA);

  par := par.AppendNewTable; // a new table
  par.ASet(WPAT_BoxWidth, WPCentimeterToTwips(9)); // 9 cm wide
  par.ASet(WPAT_BoxMarginLeft, WPCentimeterToTwips(2)); // 2 cm from left margin

  row := par.AppendNewRow(true); // the first row
  row.ASet(WPAT_PaddingAll, WPCentimeterToTwips(0.2));  // 2 mm padding

  cell := row.AppendNewCell;
  pacell.ASet(WPAT_BorderFlags, WPBRD_DRAW_All4); // all lines
  cell.ASet(WPAT_BorderWidth, WPCentimeterToTwips(0.05)); // 0,5 mm
  cell.ASetColor(WPAT_BorderColorL, clRed); // Left
  cell.ASetColor(WPAT_BorderColorT, clGreen); // Top
  cell.ASetColor(WPAT_BorderColorR, clBlue); // Right
  cell.ASetColor(WPAT_BorderColorB, clYellow); // Bottom
  cell.SetText('Cell A', RedTextA);
  cell.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(3));  // 3 cm wide

  cell := cell.AppendNewCell(false); // new cell, same colors
  cell.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(3));  // 3 cm wide
  cell.SetText('Cell B', RedTextA);

  cell := cell.AppendNewCell(false); // new cell, same colors
  cell.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(3));  // 3 cm wide
  cell.SetText('Line 1', TextA);
  cell.AppendNewPar(true).SetText('Line 2',TextA);

 // and a second row
  row := row.AppendNewRow(true); // the first row
  cell := row.AppendNewCell;
  cell.ASet(WPAT_BorderFlags, WPBRD_DRAW_All4);
  cell.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(9));
  cell.SetText('second cell', HeadlineA);
```

when done - format the text

```
  RTFMemo.DelayedReformat;
end;
```

5.3.11 Tables

WPTools 5 tables are very powerful:



A table is handled like a paragraph object which contains multiple paragraphs as children. These are the rows. In turn, the rows contain multiple paragraphs which are the cells. Each cell paragraph can contain multiple paragraphs which are either text lines or tables:

This makes WPTools tables very similar to HTML tables:

```
<table>
  <tr>
    <td>Cell 1 in Row 1</td>
    <td>Cell 2 in Row 1</td></tr>
  <tr>
    <td>Cell 1 in Row 2</td>
    <td>Cell 2 in Row 2</td></tr></table>
```

One possibility to create a table is to use the CreateTable API function:

```
var aCell : TParagraph;

with WPRichText1.Memo.DisplayedText.CreateTable(nil) do
begin
  ASet(WPAT_BorderFlags, WPBRD_DRAW_All4);
  with CreateRow(nil, true) do
  begin
    InputCell.SetText('Cell 1');
    InputCell.SetText('Cell 2');
    EndRow(ThisRowStyle)
  end;
  with CreateRow(nil, true) do
  begin
    InputCell.SetText('Cell 3');
    aCell := InputCell;
    with aCell do
    begin
      // You could create more text ort a sub table ...
    end;
    EndRow(ThisRowStyle)
  end;
end;
```

The table is created using the 'CreateTable' function of the TWPRTFDataBlock object. The parameter for this function is the paragraph after which the new table should be created. You can use 'nil' to create a table at the end of the document. The resulting value of this function is the TParagraph object which will contain all new rows. Use the CreateRow function of this paragraph to create a new row. The resulting value of the CreateRow function is a TWPTableRowStyle object, which serves as the default style for all cells created and also provides the function to create new cells. This object is deallocated by 'EndRow'.

The other possibility is to use the **API TableAdd()** with a callback function to initialize the table cells. This is usually the more powerfull way to do it. Please see next chapter for various examples.

5.3.12 Create Table in Code

Very often you will have to create a table, for example a calculation.

This can be done best using the **TableAdd** procedure. This procedure requires only a few parameters, such as row cont and column count but can also work with a callback procedure which is executed for each created cell.

Since the callback procedure receives information about the current column and row number it is easy to apply special properties to certain cells or preset the contents of the created cell.

Note: You can also use the Rows[r].Cols[c] arrays as described in the second part of this chapter but then you have to take care that you work with the correct row index. Using this arrays is probably also a little slower.

a) Use TableAdd() with a callback function

This code will append a new table to the end of the text:

```
WPRichText1.TableAdd(7,7,
   [wptblActivateBorders,wptblAppendTableAtEnd], nil, InvoiceDemoCell);
```

This is a screen shot of the created table:

	Product	Price	Amount	net	+VAT	total
1	Cool	23	23	529	84.64	613.64
2	Master	432	432	186624	29859.84	216483.84
3	Hummer	956	956	913936	146229.76	1060165.76
4	High Performace	123	123	15129	2420.64	17549.64
5	Better	55	55	3025	484	3509
						1298321.88

The callback procedure InvoiceDemoCell is implemented like this

```
procedure TWPTableCalc.InvoiceDemoCell(RowNr, ColNr: Integer; par: TParagraph);
const prods : array[1..5] of string =
 ( 'Cool', 'Master', 'Hummer', 'High Performace', 'Better' );
begin
  // Set the widths of the rows
  case ColNr of
       1 : par.ASet(WPAT_COLWIDTH_PC, 500);  // % * 100 !
```

```
2 : par.ASet(WPAT_COLWIDTH_PC, 2500);
      else
      begin
         par.ASet(WPAT_COLWIDTH_PC, 1400);
         par.ASet(WPAT_Alignment, Integer(paralRight));
      end;
  end;

  // Set the text and the cell names and commands

  if RowNr=1 then  // Header Row -----------------------------
  begin
     case ColNr of
       1 : ;
       2 : par.SetText('Product');
       3 : par.SetText('Price');
       4 : par.SetText('Amount');
       5 : par.SetText('net');
       6 : par.SetText('+VAT');
       7 : par.SetText('total');
     end;
     par.ASetColor(WPAT_FGColor, $A0A0A0);
     par.ASet(WPAT_ParProtected,1);
     par.ASet(WPAT_Alignment, Integer(paralCenter));
  end else
  if RowNr=7 then // Footer Row -----------------------------
  begin
     par.ADel(WPAT_BorderWidth); // Delete the border width for ALL linese
     par.ASet(WPAT_BorderWidthT, 40); // ANd set the top line to 40 twips
     par.ASetAdd( WPAT_BorderFlags, WPBRD_DRAW_Top);  // Add a flag!
     // par.ParentRow.ASet(WPAT_BoxMinHeight, WPCentimeterToTwips(1.5));
     par.ASet(WPAT_SpaceBefore, WPCentimeterToTwips(0.3));
     par.ASet(WPAT_SpaceAfter, WPCentimeterToTwips(0.3));
     par.ASet(WPAT_ParProtected,1);
     case ColNr of
       1 : ;
       2 : ;
       3 : ;
       4 : ;
       5 : begin
             par.ASetStringProp(WPAT_PAR_COMMAND, 'PAR_NET');
           end;
       6 : begin
              par.ASetStringProp(WPAT_PAR_COMMAND, 'PAR_VAT');
           end;
       7 : begin
              par.ASetStringProp(WPAT_PAR_COMMAND, 'PAR_TOTAL');
              par.ASetCharStyle(true, WPSTY_BOLD);
           end;
     end;

  end else  // Data Rows -----------------------------
  begin
    case ColNr of
       1 : begin
              par.SetText(IntToStr(RowNr-1));
              par.ASet(WPAT_ParProtected,1);
           end;
       2 : par.SetText(prods[RowNr-1]);
       3 : par.SetText(IntToStr(Random(1000)+1));
       4 : par.SetText(IntToStr(Random(3)+1));
       5 : begin
             par.ASetStringProp(WPAT_PAR_COMMAND, 'left(2)*left(1)');
             par.ASetStringProp(WPAT_PAR_NAME, 'PAR_NET');
             par.ASet(WPAT_ParProtected,1);
           end;
       6 : begin
              par.ASetStringProp(WPAT_PAR_COMMAND, 'left(1)*0.16');
              par.ASetStringProp(WPAT_PAR_NAME, 'PAR_VAT');
              par.ASet(WPAT_ParProtected,1);
           end;
       7 : begin
```

```
par.ASetStringProp(WPAT_PAR_COMMAND, 'left(2)+left(1)');
              par.ASetStringProp(WPAT_PAR_NAME, 'PAR_TOTAL');
              par.ASet(WPAT_ParProtected,1);
            end;
      end;
  end;
end;
```

Note: The commands

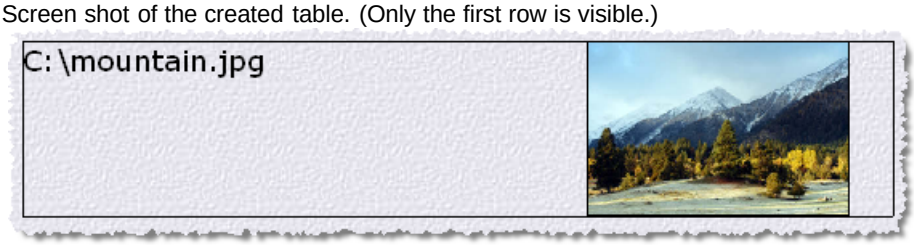
```
              par.ASetStringProp(WPAT_PAR_COMMAND, 'left(2)+left(1)');
              par.ASetStringProp(WPAT_PAR_NAME, 'PAR_TOTAL');
```

are only usable with WPTools Bundle. They are used to add calculation to a table.

b) Use TableAdd() and the Rows[] and Cols[] arrays

This example appends 2 rows to the current table (a new table is created if necessary) and loads an image in the second cell of the first row.

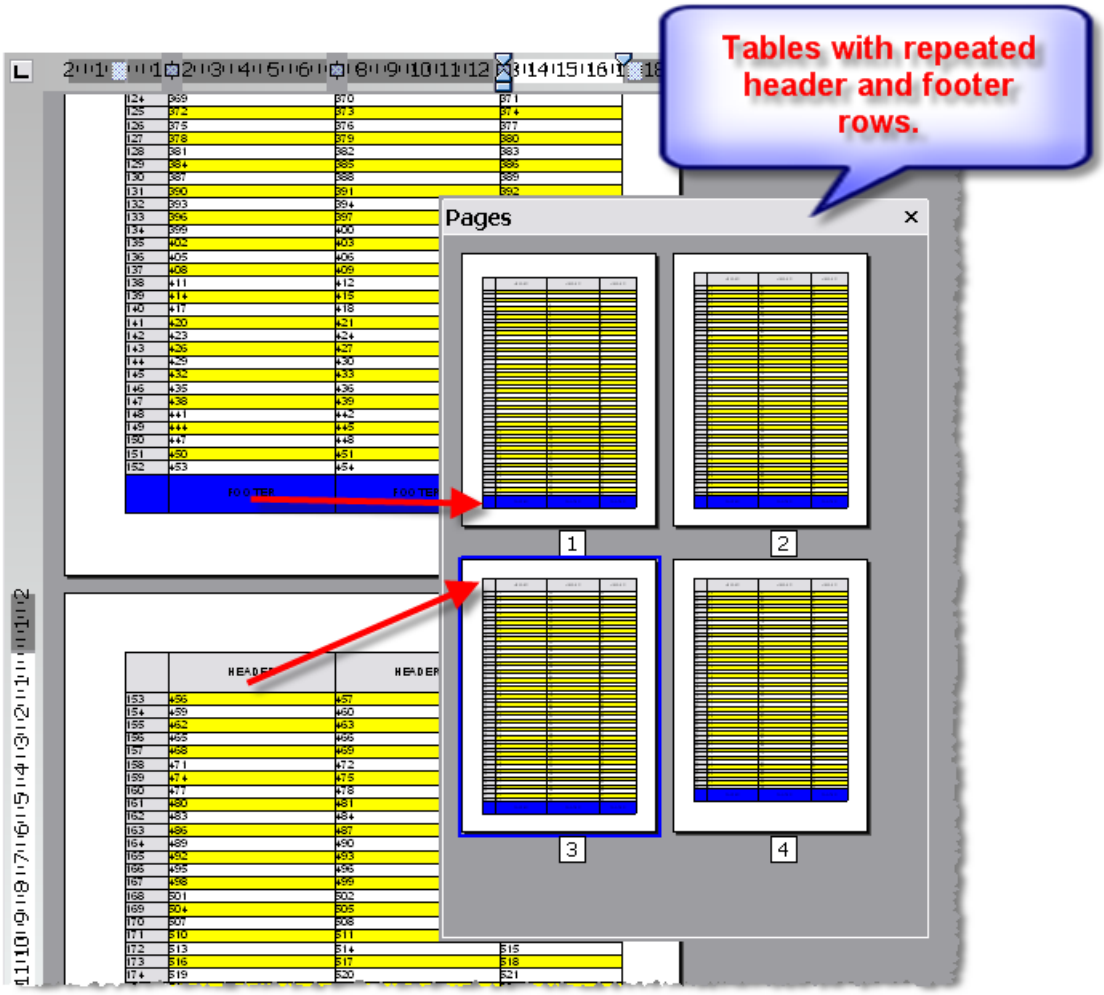
Screen shot of the created table. (Only the first row is visible.)




```
procedure TForm1.CreateTableWithImageClick(Sender: TObject);
var tbl: TParagraph;
  obj: TWPTextObj;
  img: TWPOImage;
  imagefilename: string;
  rowoffset : Integer;
begin
 // Select the image from file or use a local image
  imagefilename := '';
  if SelectImageFile.Checked then
  begin
    if OpenPictureDialog1.Execute then
      imagefilename := OpenPictureDialog1.FileName
    else exit;
  end;

 // Create the image object
  img := TWPOImage.Create(WPRichText1); // uses WPObj_Image
  try
    if imagefilename = '' then img.Picture.Assign(Image2.Picture)
    else img.LoadFromFile(imagefilename);
  except
    img.Free; // We cannot load this image
    raise;
  end;

  // This code is required if we do *not* use the wptblAppendTableAtEnd option.
  // since than an existing table is enlarged
  tbl := WPRichText1.Table;
  if tbl<>nil then
  begin
     rowoffset := tbl.RowCount;
     WPRichText1.ActiveParagraph := tbl.LastChild.ColFirst;
  end
  else rowoffset := 0;

 // Create the table and after that modify the cells
 //  makes sure a new table is created at the end!
```

```
tbl := WPRichText1.TableAdd(2, 1, [wptblActivateBorders], nil, nil);

  // Set text of first column
  tbl.Rows[rowoffset+0].Cols[0].SetText(imagefilename);

 // Create the TWPTextObj (which is the reference to image)
  obj := tbl.Rows[rowoffset+0].Cols[1].AppendNewObject(wpobjImage, false, false, 0);
  obj.ObjRef := img;
  obj.Frame := [wpframeFine];

 // Set the size of the image
  obj.Width := img.ContentsWidth * 2;
  obj.Height := img.ContentsHeight * 2;

  // Empty row, no borders - move cursor to first cell
  WPRichText1.ActiveParagraph := tbl.ColFirst;
  tbl := WPRichText1.TableAdd(2, 1, [], nil, nil);
  WPRichText1.ActiveParagraph := tbl.ColFirst;

 // Format and display changed text
  WPRichText1.Refresh;
end;
```

c) Create a table with a header and footer row which are repeated on each page



This feature is controlled by the flags paprIsFooter and paprIsHeader in the property TParagraph.par. The property must be set in the row paragraph which is the parent paragraph of a cell. The API TableAdd does this for you.

We suggest to use this feature with the flag wpfDontBreakTableRows in FormatOptions. In any case please set the property wpDisableSpeedReformat.

Example code:

```
procedure TForm1.CreateTableCellCallBackHF(RowNr, ColNr: Integer; par: TParagraph);
begin
  if RowNr = -1 then // THIS CELL IS IN THE HEADER
  begin
    par.ASetColor(WPAT_BGColor, clBtnFace);
    if ColNr = 1 then par.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(1.5))
    else par.SetText('HEADER');
  end
  else if RowNr = -2 then // THIS CELL IS IN THE FOOTER
  begin
    par.ASetColor(WPAT_BGColor, clBtnFace);
    if ColNr = 1 then par.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(1.5))
    else par.SetText('FOOTER');
  end else
  begin
    if (RowNr and 1) = 0 then par.ASetColor(WPAT_BGColor, clYellow);
    if ColNr = 1 then
    begin
      par.ASetColor(WPAT_BGColor, clBtnFace);
      par.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(1.5));
      par.SetText(IntToStr(RowNr));
    end else
    begin
      par.SetText(IntToStr(FCellNr));
      inc(FCellNr);
    end;
  end;
end;

procedure TForm1.CreateTableWithHeaderFooterClick(Sender: TObject);
begin
  WPRichText1.Clear;
  FRowCount := 200; // count of rows, excluding header and footer!
  WPRichText1.FormatOptions := [wpDisableSpeedReformat, wpfDontBreakTableRows];
  WPRichText1.TableAdd(
    4, FRowCount,
    [wptblActivateBorders,wptblCreateHeaderRow,wptblCreateFooterRow], nil,
    CreateTableCellCallBackHF);
  WPRichText1.Refresh;
end;
```

5.3.13 Work with Styles and Stylesheets (in code)

WPTools 5 has powerful possibilities to work with text styles.

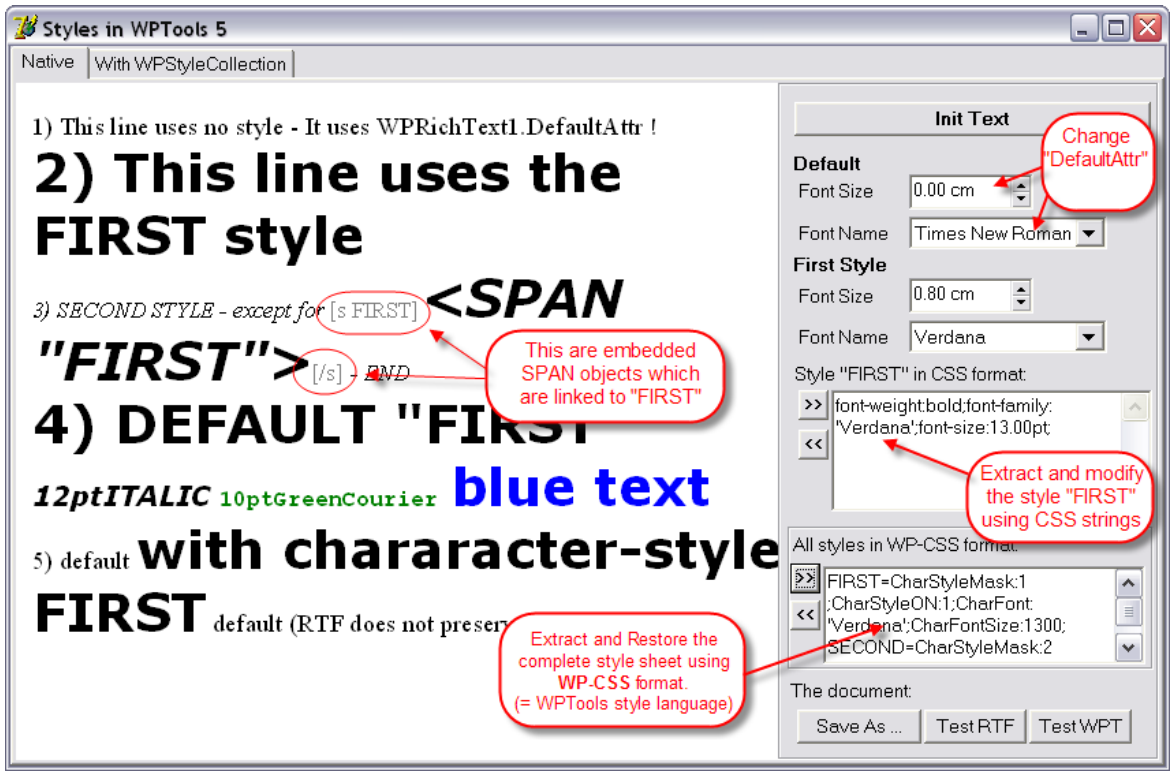
This one is one of the most important chapters of this manual since it explains how the styles - which are the heart of WPTools 5 - can be used and controlled.

1) What is a 'style'?

A text style is a collection of text attributes which are used for a certain text unless the text itself overrides the attributes. So if you have a style which defines two properties, the font name and the font size and assign it to some text this text will be displayed in the select font in the selected size. If the text was created to use the font size, ie. 10, this font size will be used and not the font size defined in the style.

Text styles usually have names, for example 'Headline' or 'Normal'. Using the names you can select a style from the list of styles and assign it to a paragraph. This assignment usually does not modify the attributes of the text. (NOTE A)

2) The demo project (Demos/Tasks/ParStyle**):**



This demo shows different ways to deal with the styles. You can play with the default attribute and the properties of one if the styles (FIRST) and immediately see the effect.

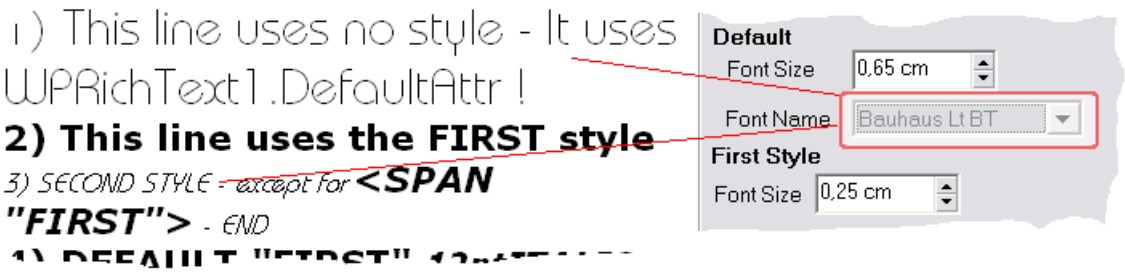
3) The default attribute

This attribute (or elements of it) will be used for all the text which does not have an attribute ion its own.

In this demo we initialize the default attribute using:

```
WPRichText1.DefaultAttr.SetFontName(ComboBox1.Text);
WPRichText1.DefaultAttr.SetFontSize(WPValueEdit2.Value  / 20);
```

While the demo is running you can change the defined values and immediately see the effect. So all the paragraph styles (FIRST and SECOND) set their own font size - changing the default will not change the display. But if you are changing the font name, you will see the change not only in the first line but also in the 3rd line:



So how is text created which only uses the default attribute?

```
par := WPRichText1.ActiveText.AppendPar;
par.SetText('1) This line uses no style - It uses WPRichText1.DefaultAttr !');
```

But in general, when you write text into the editor and no specific font was selected - that entry will be 'undefined' and the default will be used.

4) Creating a paragraph style

In this demo we create a style (`sty : TWPTextStyle;`) like this:

```
sty := WPRichText1.ParStyles.AddStyle( 'FIRST');
```

and then assign properties using the ASet methods:

```
sty.ASetAddCharStyle(WPSTY_BOLD);
sty.ASetFontName('Verdana');
sty.ASet(WPAT_CharFontSize, 13 * 100);
```

Many customers have asked why the styles do not have properties, such as sty.FontName? The reason for this is that using functions to read and write the properties we can support the <undefined> state. This means that if a font name was not set in a style we can react on that (by using the inherited font name). In general are the ASet... methods used to write a property while the AGet function are used to read it. They usually require a var parameter which will be changed. The result value of an AGet function is TRUE if the property was defined, otherwise it is FALSE. There is also the AGetDef() function which returns the property value itself, if it was not found it returns the default which was passed as parameter. (Read more about ASet)

To use the paragraph style we can set the name

```
par.ABaseStyleName := 'FIRST';
```

or the style:

```
par.ABaseStyle := sty;
```

5) Save and restore style sheet in WPCSS format

```
// Save:
   Memo1.Text := WPRichText1.ParStyles.GetWPCSS;
//or
   WPRichText1.ParStyles.SaveToFile( 'c:\stylesheet.wpcss');
// Load:
   WPRichText1.ParStyles.SetWPCSS(Memo1.Text);
   WPRichText1.ReformatAll(true,true);
//or
   WPRichText1.ParStyles.LoadFromFile( 'c:\stylesheet.wpcss');
   WPRichText1.ReformatAll(true,true);
```

6) Creating embedded SPAN objects (Line 3)

Especially to provide strong HTML support WPTools 5 can work with embedded SPAN styles. This are special objects which are embedded into the text. They are always used in pairs, one opening and one closing. The opening object can select a style and the closing object selects the previous settings again.

If you are working with RTF you should not use SPAN objects.

In this demo we create the objects right in the TParagraph object. If you need a less abstract way to work with the objects please check out the 'Code' procedures implemented in TWPRichText. They make it easy to create the objects, for example wrap selected text into a pair of SPAN tags.

```
par := WPRichText1.ActiveText.AppendPar;
pos := par.Insert(0, '3) SECOND STYLE - except for ',0);
spanobj_open := par.InsertNewObject(pos,wpobjSPANStyle, true, false);
spanobj_open.StyleName := 'FIRST';
inc(pos);
pos := par.Insert(pos, '<SPAN "FIRST">',0);
spanobj_close := par.InsertNewObject(pos,wpobjSPANStyle, true, true);
spanobj_close.SetTag(spanobj_open.NewTag); //<-- tags are used to link start with end!
inc(pos);
pos := par.Insert(pos, ' - END',0);
par.ABaseStyleName := 'SECOND';
```

The variables *spanobj_open* and *spanobj_close* are of type TWPTextObj. They are created by InsertNewObject which requires four parameters:

```
index: Integer;
objtype: TWPTextObjType;
HasClosing, IsClosing: Boolean
```

Please note how the value for 'IsClosing' is alternated the the demo source code. "pos" is an integer value which is the index into the paragraph. The insert() function always returns the position after the inserted text, so pos is always incremented.

Please note, that for each open SPAN object the closing object must be on the same paragraph. If you need a linebreak you can use the code Char(10) to create a soft line break.

Usually SPAN objects are not visible. They can be made visible using the flag **wpShowSPANCodes** in property Formatoptions. The display is controlled by property SPANObjectTextAttr, the text can be changed using SPANObjectTextAttr.CodeOpeningText and SPANObjectTextAttr.CodeClosingText (%Y inserts the StyleName).

7) Working with the Character Attributes (Line 4)

In this demo we wanted to show how to use the low level methods. You can of course also create text using the procedure InputString and modify the current writing mode using the interface 'CurrAttr' - the demo GridMode uses the InputString technology. But this demo creates the text a bit differently. **Please don't read on if you find this confusing**. Using the AttrHelper interface ist usually much easier.

First we need a buffer for our character attributes:

```
ca : TWPCharAttr;
```

Then we create the paragraph:

```
par := WPRichText1.ActiveText.AppendPar;
```

The buffer is initialized with

```
FillChar(ca, SizeOf(ca), 0);
```

If you are using Delphi 8 you can define another `emptyca : TWPCharAttr` and assign it to clear ca.

Now we fill the paragraph by appending text.

```
  pos := par.Insert(0,'4) DEFAULT "FIRST" ',0);
  par.RTFProps.AttrInterface.SetCharStyles(ca, WPSTY_ITALIC, WPSTY_ITALIC);
  par.RTFProps.AttrInterface.SetFontSize(ca, 12);
  pos := par.Insert(pos,'12ptITALIC ',ca);
```

"pos" is an integer value which is the index into the paragraph. The insert() function always returns the position *after* the inserted text, so *pos* is always incremented.

The lines with `par.RTFProps.AttrInterface....` are used to modify ***ca*** - which in the end will be used as parameter to insert().

AttrInterface is an object of class TWPCharAttrInterface. This class is not used to store attributes but to modify buffer variables of type TWPCharAttr!

8) The complete source of the "Init Text" procedure

```
var par: TParagraph;
  sty: TWPTextStyle;
  pos: Integer;
  spanobj_open, spanobj_close: TWPTextObj;
  ca: TWPCharAttr;
begin
  WPRichText1.DefaultAttr.SetFontName(ComboBox1.Text);
  WPRichText1.DefaultAttr.SetFontSize(WPValueEdit2.Value / 20);

  WPRichText1.Clear;
  // Set BODY
  WPRichText1.ActiveText := WPRichText1.BodyText;
  // ------------------------------------------------- FIRST LINE -----------
  // We use the DefaultAttr -----------------------------------------------
  par := WPRichText1.ActiveText.AppendPar;
  par.SetText('1) This line uses no style - It uses WPRichText1.DefaultAttr !');
  // ------------------------------------------------- SECOND LINE ----------
  // We use the default Style for the complete paragraph ------------------
  // Creates Style
  sty := WPRichText1.ParStyles.AddStyle('FIRST');
  // Create Paragraph
  par := WPRichText1.ActiveText.AppendPar;
  par.SetText('2) This line uses the FIRST style');
  // Assign Style
  par.ABaseStyle := sty;
  // Set Props
  sty.ASetAddCharStyle(WPSTY_BOLD);
  sty.ASetFontName('Verdana');
  sty.ASet(WPAT_CharFontSize, 13 * 100);
  // ------------------------------------------------- THIRD LINE -----------
  // We embedd a SPAN style -----------------------------------------------
  // Creates Style
  sty := WPRichText1.ParStyles.AddStyle('SECOND');
  // Set Props
  sty.ASetAddCharStyle(WPSTY_ITALIC);
  sty.ASet(WPAT_CharFontSize, 10 * 100);
  // Create Paragraph
  par := WPRichText1.ActiveText.AppendPar;
  pos := par.Insert(0, '3) SECOND STYLE - except for ', 0);
  spanobj_open := par.InsertNewObject(pos, wpobjSPANStyle, true, false);
  spanobj_open.StyleName := 'FIRST';
  inc(pos);
  pos := par.Insert(pos, '<SPAN "FIRST">', 0);
  spanobj_close := par.InsertNewObject(pos, wpobjSPANStyle, true, true);
  spanobj_close.SetTag(spanobj_open.NewTag); //<-- tags are used to link start with end!
  inc(pos);
  par.Insert(pos, ' - END', 0);
  par.ABaseStyleName := 'SECOND';
  // ------------------------------------------------- 4th LINE -------------
  // We use character attributes ------------------------------------------
  // 'ca' is used as buffer to create a char attribute
  par := WPRichText1.ActiveText.AppendPar;
```

```
FillChar(ca, SizeOf(ca), 0);
  pos := par.Insert(0, '4) DEFAULT "FIRST" ', 0);
  par.RTFProps.AttrInterface.SetCharStyles(ca, WPSTY_ITALIC, WPSTY_ITALIC);
  par.RTFProps.AttrInterface.SetFontSize(ca, 12);
  pos := par.Insert(pos, '12ptITALIC ', ca);

  FillChar(ca, SizeOf(ca), 0);
  par.RTFProps.AttrInterface.SetFontName(ca, 'Courier New');
  par.RTFProps.AttrInterface.SetFontSize(ca, 10);
  par.RTFProps.AttrInterface.SetColor(ca, clGreen);
  pos := par.Insert(pos, '10ptGreenCourier ', ca);

  FillChar(ca, SizeOf(ca), 0);
  par.RTFProps.AttrInterface.SetColor(ca, clBlue);
  par.Insert(pos, 'blue text', ca);

  // Assign the FIRST Style
  par.ABaseStyleName := 'FIRST';

  // ---------------------------------------------------  last LINE --------
  // We use a character style -----------------------------------------------
  // 'ca' is used as buffer to create a char attribute
  par := WPRichText1.ActiveText.AppendPar;

  FillChar(ca, SizeOf(ca), 0);
  pos := par.Insert(0, '5) default ', 0);
  par.RTFProps.AttrInterface.SetCharStyleSheet(ca,
           par.RTFProps.ParStyles.GetID('FIRST'));
  pos := par.Insert(pos, 'with chararacter-style FIRST', ca);
  par.Insert(pos, ' default (RTF does not preserve character styles)', 0);
end;
```

9) Working with CSS

CSS means Cascading Style Sheet and is used to store any kind of paragraph and character attributes in HTML files. WPTools 5 was created keeping in mind the philosophy of both, CSS and RTF to provide the developer the best of both worlds.

[>>] WPTools' TWPTextStyle class has the ability to dump its properties (to be exact: only the properties supported by CSS) as a CSS string:

```
var sty : TWPTextStyle;
begin
   sty := WPRichText1.ParStyles.FindTextStyle('FIRST');
   if sty<>nil then
      CSS1.text := sty.AGet_CSS(true,false,true);
end;
```

[<<] To load a CSS string back into a TWPTextStyle you can use the TWPCSSParserStyleWP which is implemented in unit WPIOCSS:

```
var sty : TWPTextStyle;
    parser : TWPCSSParserStyleWP;
begin
   sty := WPRichText1.ParStyles.FindTextStyle('FIRST');
   if sty<>nil then
   begin
      parser := TWPCSSParserStyleWP.Create;
      try
         parser.AsString := CSS1.text;
         parser.ApplyToStyle(sty);
         WPRichText1.ReformatAll(true); // Initialize all because we changed the font
         WPRichText1.Repaint;
      finally
         parser.Free;
      end;
   end;
end;
```

Please note that the TParagraph inherits from TWPTextStyle - so the above techniques can be also used for any paragraph object!

10) Working with WPCSS

WPTools can work with string representation of property settings. Theses are called WPCSS strings due to their similarity to CSS.

To read a WPCSS string use

```
function TWPTextStyle.AGetWPSS(
      Names,
      CharAttr,
      TabAttr: Boolean;
      OnlyUsePTag: Boolean = FALSE;
      Abbreviated: Boolean = FALSE): AnsiString;
```

To write it use

```
procedure TWPTextStyle.ASetWPSS(
      const WPCSSString: AnsiString;
      Merge: Boolean = false;
      Abbreviated: Boolean = FALSE);
```

If OnlyUsePTag is set to TRUE no style names are written. You should not use this feature if you want to store the created string. It will be invalid the next time the text is created.

If Abbreviated is TRUE the short form for the property names will be used.

5.3.14 Numbering

In WPTools 5 you can create paragraphs with bullets, simple numbering our outline numbering.

The numbering is always controlled by the collection NumberStyles which is accessible by TWPRichText.NumberStyles.

Each entry in this collection has a unique ID which is selected in each numbered paragraph using the WPAT_NumberStyle property. In case of outline numbering this style id only needs to select one style from the correct group. The engine will then use the outline level (WPAT_NumberLevel) to select the style within the same group.

If a paragraph uses a paragraph style the NumberStyle defined in this style (if any) has priority. The number level of a paragraph has priority over the number level defined by a paragraph style!

Please note: The properties WPAT_NumberMode, NumberTEXTB, NumberTEXTA etc should NOT be used for paragraphs or paragraph styles. They are only handled by the styles which are part of NumberStyles collection.

Example: Initialize the NumberStyles to use legal numbering 1, 1.1, 1.2 ...

```
var
  i: Integer;
begin
  if WPRichText1 <> nil then
    for i := 1 to 9 do
      with WPRichText1.NumberStyles.AddOutlineStyle( 1, i) do
      begin
        Style := wp_1;
        TextB := '';
        TextA := '.';
        Font := '';
        Indent := 360 * i;
        UsePrev := TRUE;
      end;
    WPRichText1.Refresh;
end;
```

Example using 'SelectedTextAttr': Assign simple 1,2,3 numbering to the selected text:

```
var ind : Integer;
begin
  ind := 360;
  WPRichText1.SelectAll;
  WPRichText1.SelectedTextAttr.ASet(
     WPAT_NumberStyle,
     WPRichText1.NumberStyles.AddNumberStyle(
       wp_1, // possible values: wp_bullet, wp_circle,
             // wp_1, wp_lg_i, wp_i, wp_lg_a, wp_a,
       Before1.Text,
       After1.Text,
       '', // Font, important for bullets
       ind, // default indent
       0, // Fontsize or default
       false, // = legal numbering
       0, // group, 1 for outline numbering
       0  // level in group 1..10
     ));
end;
```

If the selected numbering style is an outline style, you can use ASet(WPAT_NumberLevel, level) to select the style within this level.

This example will apply the level one of the default outline group to either the current paragraph or the selected text if any selection has been made.

```
var sty : TWPRTFNumberingStyle;
begin
  // Locate level 1 in default outline group
  sty := WPRichText1.NumberStyles.FindNumberStyle(-1,1);
  // and assign the ID to the current paragraph or the selected
  // paragraphs
  WPRichText1.ASet(WPAT_NumberSTYLE, sty.ID);
  WPRichText1.ASet(WPAT_NumberLEVEL, 1);
  WPRichText1.Refresh;
end;
```

Like the paragraph styles the elements in "NumberStyles" have a property 'TextStyle'. This is a standard TWPTextStyle instance which holds the attributes for the numbering level. You can use it to set additional properties, such as the font color for the numbers.

Example using 'CurrAttr': Assign simple 1,2,3 numbering to the selected text:

```
var ind : Integer;
begin
  ind := 360;
  WPRichText1.SelectAll;
  WPRichText1.CurrAttr.BeginUpdate;
  WPRichText1.CurrAttr.IndentLeft := ind;
  WPRichText1.CurrAttr.IndentFirst:= -ind;
  WPRichText1.CurrAttr.SetNumberStyle(
     Before1.Text, After1.Text, '', wp_1, ind );
  WPRichText1.CurrAttr.EndUpdate;
  WPRichText1.HideSelection;
end;
```

Example using 'CurrAttr': Create several outline levels in the text

```
var cp : Integer;  i,l,m : Integer;
begin
  cp :=  WPRichText1.TextCursor.DropMarker;
  // Remove numbers and indent
  WPRichText1.SelectAll;
  WPRichText1.CurrAttr.BeginUpdate;
  WPRichText1.CurrAttr.NumberStyle := 0;
  WPRichText1.CurrAttr.IndentLeft := 0;
  WPRichText1.CurrAttr.EndUpdate;
  WPRichText1.HideSelection;
  i := 0;
  l := 1;
  m := 1;
  // Move down and apply Outline Mode
  WPRichText1.CPPosition := 0;
  repeat
    WPRichText1.CurrAttr.BeginUpdate;
    WPRichText1.CurrAttr.OutlineLevel := l;
    WPRichText1.CurrAttr.IndentLeft := l * 360;
    WPRichText1.CurrAttr.IndentFirst := -360;
    WPRichText1.CurrAttr.EndUpdate;
    inc(i);
    if i=3 then
    begin
      l := l+m;
      if (l=4) or (l=1) then m := -m;
      i := 0;
    end;
  until not WPRichText1.CPMoveDownPar;
end;
```

Note about CurrAttr: This is an interface which was introduced in WPTools 3. It is still supported and implemented in unit WPCTRRich. If you do not want to link unit WPCTRRich you cannot use CurrAttr. For new code we suggest to use ActiveParagraph, ASet or SelectTextAttr methods.

5.3.15 Hyperlinks

Certain texts can be interactive in WPTools - this means the text can be automatically highlighted when the mouse is moved over it (hover or "hot" effect) or an event can be triggered when the user clicks on it.

The standard interactive texts are **hyperlinks**.

Hyperlinks in WPTools 5 start with a hyperlink start tag <a> and end with a closing tag . All this tags are represented by TWPTextObj instances. (These tags can be made visible using the 'FormatOptions'!)

This code can be used to create a hyperlink:

```
WPRichText1.InputHyperlink('WPTOOLS','BOOK1').
```

Since hyperlinks are marked with paired TWPTextObj instances it is also possible to create those tags directly by code. This is very useful if you are creating the text using the low level paragraph procedures:

```
var par : TParagraph;
    link_tag : TWPTextObj;
begin
 par := WPRichText1.ActiveText.AppendNewPar();
 par.Append('This is a link: ');                              // Some Text
 // Create a link
 link_tag := par.AppendNewObjectPair(wpobjHyperlink,'WPCubed GmbH');
 link_tag.Source := 'http://www.wpcubed.com';
 // to display:
 WPRichText1.Refresh;
end;
```

Note: The above code uses the function AppendNewObjectPair´. This function internally executes code similar to this example:

```
startobj := par.AppendNewObject(wpobjHyperlink,true,false);  // Opening
par.Append('WPCubed GmbH');          // some text
endobj   := par.AppendNewObject(wpobjHyperlink,true,true);   // Closing
endobj.SetTag(startobj.NewTag);      // Link Opening<->Closing
startobj.Source := 'http://www.wpcubed.com' ;                // The URL
```

Hyperlink tags can have a style attached. This style can be a paragraph style:

```
startobj.StyleName := ParStyleName; // set a reference to a paragraph style
```

If no style name was specified the list of paragraph styles will be searched for a style with the name "A".

Alternatively the style can be defined directly in the TWPTextObj instance:

```
startobj.MakeStyle(true);
startobj.Style.ASet(WPAT_CharFontSize, 2200);     // create a large font
startobj.Style.ASetColor(WPAT_CharColor, clRed);  // as red text
startobj.Style.ASetColor(WPAT_CharBGColor, clYellow); // on yellow background
startobj.Style.ASetFontName('Courier New');
```

Please note that these style definitions have a lower priority than the text attributes defined for the enclosed characters (this are the attributes which have been assigned to the text) and the attributes defined in the property AutomaticTextAttr. They are only loaded and saved in the WPTOOLS format, not in RTF format.

React on the click on hyperlinked text: Event OnClickHotText

```
procedure TForm1.WPRichText1HyperLinkEvent(Sender: TObject; text,
  url: String; IgnoredNumber: Integer);
begin
  if Pos('http:',url)>0 then
    ShellExecute(Handle, 'open', PChar(url), '', '', SW_SHOW       );
end;
```

This event is executed after a dounble click on hyperlinked text. It will be triggered after a single

click if the property 'OneClickHyperlink' has been set to true.

React on the click on hyperlinked text: Event OnClickHotText

```
procedure TForm1.WPRichText1ClickHotText(Sender: TObject; par: TParagraph;
  posinpar, X, Y: Integer; Button: TMouseButton; Shift: TShiftState;
  TxtObj: TWPTextObj);
begin
  if TxtObj.ObjType=wpobjHyperlink then
  begin
    WPRichText1.BookmarkMoveTo(TxtObj.Source);
  end;
end;

this bookmark was created with WPRichText1.BookmarkInput('BOOK1');
```

Which types of text objects are clickable is always controlled by property **ClickableCodes**. This means the event **can be also created** for text which has been wrapped by **merge fileds, bookmarks or span codes**. The event OnClickHotText is always triggered by a single click.

Please note that in WPTools 5 hyperlinks are not represented by text with a certain attribute (asfHyperlink) followed by hidden text (afsHidden) as it was in WPTools 4 and before!

How to display visited hyperlinks in a different color:

We use the property
```
WPRichText1.HyperlinkTextAttr.UseOnGetAttrColorEvent := TRUE;
```

and a string list:
```
FVisitedHyperlinks := TStringList.Create;
```

In the hyperlink event we add URLs to the stringlist to know later if the link was already clicked:

```
procedure TForm1.WPRichText1HyperLinkEvent(Sender: TObject; text,
  url: string; IgnoredNumber: Integer);
begin
  FVisitedHyperlinks.Add(url);
  if Pos('www', url) > 0 then
    ShellExecute(Handle, 'open', PChar(url), '', '', WM_SHOWWINDOW);
end;
```

Now we can use the OnGetAttributeColor event to modify the special text attribute 'on the fly':

```
procedure TForm1.WPRichText1GetAttributeColor(Sender: TObject;
  var CharStyle: TCharacterAttr; par: TParagraph; posinpar: Integer);
var txtobj : TWPTextObj;
begin
  txtobj := WPRichText1.CodeInsideOf(par,posinpar,wpobjHyperlink);
  if txtobj<>nil then
  begin
     if FVisitedHyperlinks.IndexOf(txtobj.Source)>=0 then
          CharStyle.TextColor := clGreen;
  end;
end;
```

The reference `CharStyle: TCharacterAttr` which is passed to the event is a reference to a copy of the HyperlinkTextAttr property object. We can modify it without any sideffect to the other links in the text. But please do not modify the reference, just the properties of this object!

This technique can be also used for other 'special texts' - but the property UseOnGetAttrColorEvent must be always set to true to activate the feature.

5.3.16 Hover Effects

Certain text can change the attributes when the mouse is moved over it. In addition a hint window can be displayed. This is useful for mail merge fields and hyperlinks.

All text which support this kind of interaction can also use global style and color definitions set up in these properties:

```
i)   property HyperlinkTextAttr: TCharacterAttrTags;
     property BookmarkTextAttr: TCharacterAttrTags;
     property SPANObjectTextAttr: TCharacterAttrTags;
     property AutomaticTextAttr: TCharacterAttr;

ii)  property ProtectedTextAttr: TCharacterAttr;
     property HiddenTextAttr: TCharacterAttr;

iii) property FieldObjectTextAttr: TCharacterAttr ;
     property InsertPointAttr: TCharacterAttrTags;
```

The group i) defines attributes for texts which is wrapped in TWPTextObj instances of the following types: wpobjHyperlink, wpobjBookmark, wpobjSPANStyle and wpobjMergeField (= merged text).

The two prioperties in group ii) affects text which uses the character style afsProtected and afsHidden.

In group iii) FieldObjectTextAttr changes the appearance of wpobjTextObject objects (with the exception of objects with the name 'PAGE', 'NUMPAGES', 'SYMBOL'.
InsertPointAttr changes the way wpobjMergeField objects are displayed and can be used to hide those objects.

The **TCharacterAttr** class contains various properties to change the color of the "special" text, to add or remove underlines and to set the underline color.

To display hint windows two events can be used:

a) the OnActivateHint event

This event is triggered when the mouse is moved over special text which uses the property OnHintEventIsActive set to true in the respective TCharacterAttr property. Since there is no OnDeactivateHint event we suggest to use a timer to hide the window. In contrast to property "HotStyleIsActive" the property "OnHintEventIsActive" does not force a repaint of the text window!

a) the OnActivatingHotStyle event

This event is triggered when the mouse is moved over special text which uses the property HotStyleIsActive set to true in the respective TCharacterAttr property.

This code can be used to show a hint window with information about merged text. HotStyleIsActive must be set to TRUE in property AutomaticTextAttr:

```
procedure TForm1.WPRichText1ActivatingHotStyle(Sender: TObject;
  par: TParagraph; posinpar: Integer);
var p : TPoint;
begin
  if par <> nil then
  begin
     FHintForm.Caption := (Sender as TWPCustomRTFEdit).FieldGetNameInPar(par,posinpar);
p := TWPCustomRTFEdit(Sender).GetPointFromParLin(par, posinpar);
     if p.x > TWPCustomRTFEdit(Sender).Width then p.x := TWPCustomRTFEdit(Sender).Width;
     p := TWPCustomRTFEdit(Sender).ClientToScreen(p);
     FHintForm.Left := p.x;
```

```
FHintForm.Top := p.y;
     FHintForm.Show;
  end;
end;
```

The hint form is hidden in event OnDeactivateHotStyle

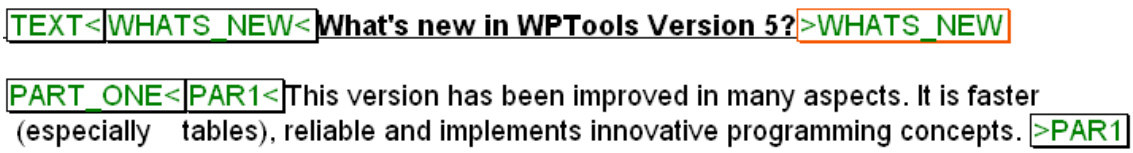
```
procedure TForm1.WPRichText1DeactivateHotStyle(Sender: TObject);
begin
   FHintForm.Hide;
end;
```

If you need to use OnClick events for certain texts use the event OnClickHotText and the property ClickableCode. See previous chapter for more information.

5.3.17 Bookmarks

The bookmarks of WPTools 5 are very powerful. They can be nested and several functions make it easy to work with the marks.

It is possible tio hide the bookmarks or to sow them. When they are displayed it is even possible to display themy through owner drawn code as in this example:



This code creates a book mark

```
  WPRichText1.BookmarkInput('BM' + IntToStr(bm),true);
```

The following functions deal with bookmarks

```
//:: finds and selects a Bookmark
    function BookmarkSelect(const Name: string; OnlyText: Boolean): Boolean;
    //:: locates a bookmark and and moves Cursor
    function BookmarkMoveTo(const Name: string): Boolean;
    //:: locates a bookmark and and moves Cursor. Start at the current position
    function BookmarkMoveToNext(const Name: string): Boolean; // locates and moves Cursor
to next ..
    //:: locates and returns position (or -1 iof not found)
    function BookmarkFind(const Name: string): Integer;
    {:: Creates a bookmark. If text is currently selected the selected text
       will be put iside of the new bookmark markers }
    function BookmarkInput(const Name: string; PlaceCursorBetweenTags: Boolean =
FALSE):TWPTextObj;
    procedure BookmarkDeleteAllMarkers;
    {:: Deletes the bookmark markers with the given name. This procedure
    searches through all blocks of the current text }
    procedure BookmarkDeleteMarkers(const Name: string);
    procedure BookmarkDelete(const Name: string; Marks, Text: Boolean);
    function BookmarkDeleteInPar(const NameStart: string; par: TParagraph): Boolean;
    procedure BookmarkGetList(list: TStrings; FromAllBlock: Boolean = FALSE); overload;
    procedure BookmarkGetList(list: TWPTextObjList; FromAllBlock: Boolean = FALSE);
overload;

    {:: This functions checks if the text at the given position is locate
    inside of a bookmark.<br>
```

```
To check if there is a bookmark object under the mouse
    cursor use CodeAtXY(X,Y,Code)! }
    function BookmarkAtXY(x, y: Integer; var Bookmark: TWPTextObj): Boolean;
    {:: This functions checks if the cursor is inside of a bookmark.<br>
    To check for a text object, such a bookmark object at the cursor position
    use CPAttr.GetObject  }
    function BookmarkAtCP: TWPTextObj;
    function BookmarkFirstInPar(par: TParagraph): string;
    function BookmarkAtParLin(par: TParagraph; pos_in_par: Integer): TWPTextObj;
    function BookmarkForceInPar(par: TParagraph; const frmstr: string): string; //
reserved
```

Example:

This code in the event handler for **OnTextObjGetTextEx** is used to paint the bookmarks in the image above:

```
procedure TForm1.WPRichText1TextObjGetTextEx(RefCanvas: TCanvas;
  TXTObject: TWPTextObj; var PrintString: WideString; var WidthInPix,
  HeightInPix: Integer; var PaintObject: TWPTextObj; Xres, YRes: Integer);
begin
  if not ShowBookmarkCodes.Checked then exit;
  if TXTObject.ObjType=wpobjBookmark then
  begin
     // If you set PaintObject to anything <> nil PrintString will be ignored!
     // Here you can initialize an event for an owner draw object
     PaintObject := TXTObject;
     PaintObject.OnPaint := OnPaintObject;
     WidthInPix := RefCanvas.TextWidth(TXTObject.Name+'<'+#32);
  end;
end;

procedure TForm1.OnPaintObject(Sender : TWPTextObj;
    OutCanvas  : TCanvas;
    XRes, YRes : Integer;
    X, Y, W, H, BASE: Integer );
var s : string;
begin
    OutCanvas.Rectangle(x,y,x+w,y+h);
    OutCanvas.MoveTo(x+w,y+1);
    OutCanvas.LineTo(x+w,y+h);
    OutCanvas.LineTo(x+1,y+h);
    if wpobjIsClosing in Sender.Mode then
       s := '>' + Sender.Name
    else s := Sender.Name + '<';
    OutCanvas.TextOut(x+2,y+BASE,s);
end;
```

Sometimes you will need to **replace the text which is wrapped within bookmarks**. This can be useful to process mailmerge forms which used to be processed using a word processor and OLE.

The following code is fast and effective:

```
procedure TForm1.ReplBookmarksClick(Sender: TObject);
var
  allbm : TWPTextObjList;
  i : Integer;
begin
   allbm := WPRichText1.CodeListTags(wpobjBookmark,'*ALL*',true);
   try
     for i:=0 to allbm.Count-1 do
     if allbm[i].Name='sum' then // check the name!
     begin
       // simply assign text. You can even create new paragraphs
       // with #13 or #13+#10 sequences.
       allbm[i].EmbeddedText :=
          '$134.39'+#13#10
         +'$180.00'+#13#10
         +'$200.00'+#13#10+'$272.00';
```

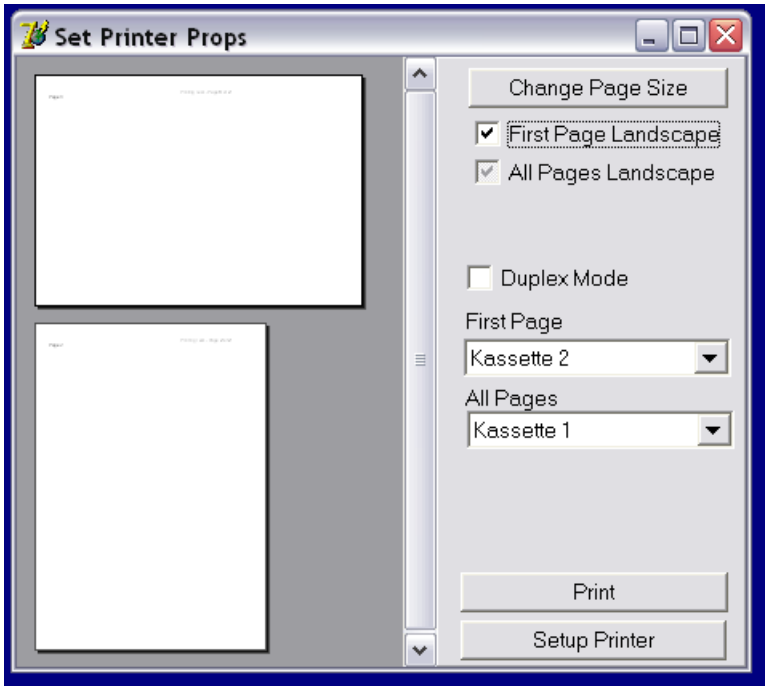
```
end;
   finally
     allbm.Free;
   end;
   WPRichText1.ReformatAll(false,true);
end;
```

5.3.18 Printing - how to set printer properties

WPTools 5 provides you with the possibility to easily change important printer options (such as paper tray) using an easy to use property PrintParameter.
During the printing the "PrintParameter" are evaluated and applied to the DEVMODE structure which is used by the printer driver. There is also the event *OnSetupPrinterEvent* to let you change the DEVMODE structure yourself. This feature can be disabled using the flag **wpDoNotChangePrinterDefaults in property PrintParameter.PrintOptions.**

The demo 'PrinterSet' shows how to:

- change the orientation of a certain page
- activate duplex mode (requires support of the printer!)
- display the available paper trays
- change the paper tray for the first and the rest of the pages
- print the document
- use BeginPrint/EndPrint to print the contents of two different editors into one printer cue - with a continuous page numbering.



You can use the TWPRichText method Print to print the complete document or PrintPages to print a range of documents. If you first call **BeginPrint(title, pagenumber)** a new printing cue is opened. For best results it is necessary to provide this function with the number of the first page. This number is 0 based. (This is necessary since the printer properties for the first page must be

set before Printer.BeginDoc is executed)

If you print from different editors please make sure that the text in all this editors is formatted. If an editor is not visible call ReformatAll! To get the correct page numbering assign the total page count to WPRichText. **Enviroment.CombinedPrintPageCount**. You will also need to set the flag **wpUsePrintPageNumber** in the property PrintParameter.PrintOption of the editor which started the printing cue. The value of this -first- property will automatically be used by all the editors which are printing into the same printing cue.
When printing is finished don't forget to call EndPrint!

Notes:

- The property Printer.PageNumber is not updated as usual.
- We recommend to add a possibility (i.e. INI entry) to your application to make it possible for the end user to set wpDoNotChangePrinterDefaults.
- WPTools 5 selects the paper from the list of the available papers of this printer. Only if the paper is not found (by comparing the size) 'custom' is used. The list of papers is updated at printime, property PageDefs is not used. So it is possible to change the printer (using Printer.PrinterIndex) at any time!
- Please also see the chapter about WYSIWYG
- You will need to temporarily hide mail-merge markers using the propertry InsertPointTextAttr.Hidden if this flag is not true anyways.

5.3.19 Superprint: Print Booklets and Labels

We've added a new component, TWPSuperPrint, that allows you to centrally manage a variety a WPTools printing features. Should you wish to, you also have the option of bypassing the Print Console, and controlling these aspects directly within your code, hidden from the user.

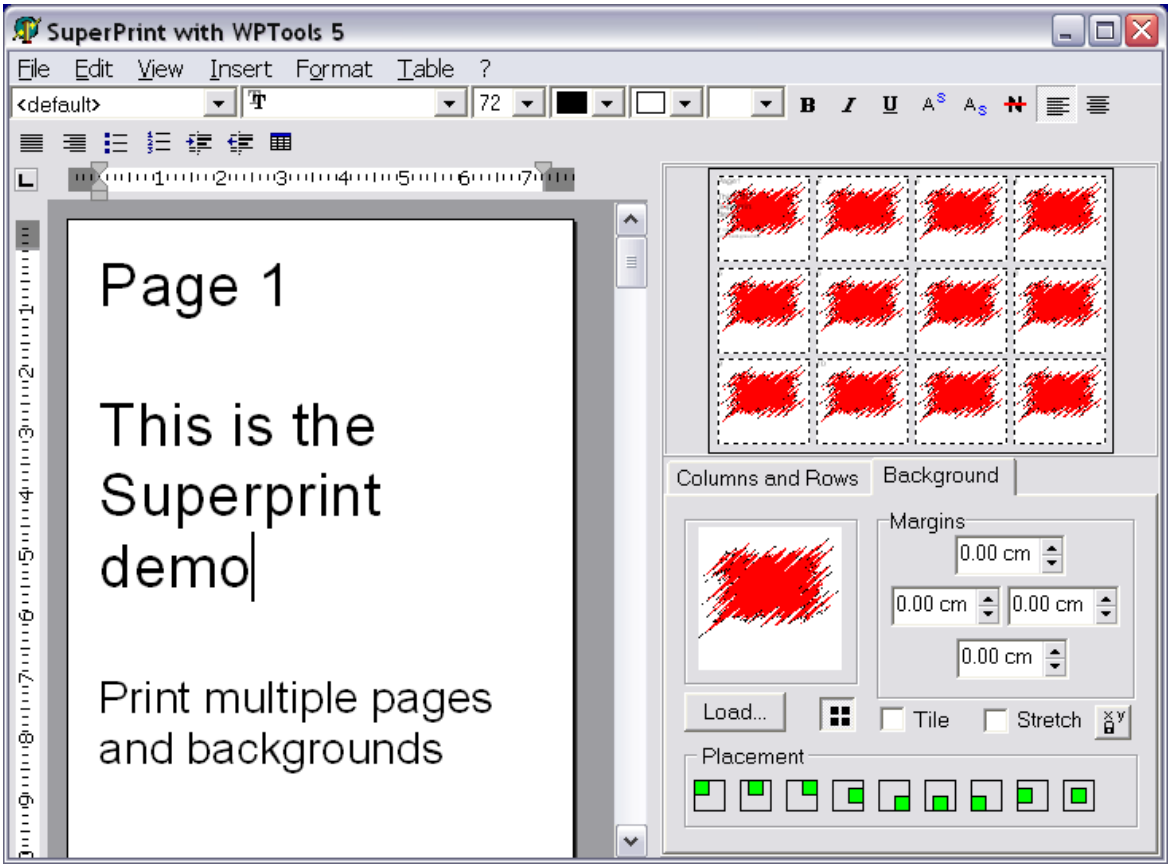
This component includes the ability to control:

- true "booklet-style" printing – the component has been optimized to automatically control "two up" (two pages per sheet) booklet printing
- printing of labels (automatically tiled across the page! See other demo)
- include a background image, selected via a subdirectory browser
- tiling or stretching of the included background image
- placement of the background image either vertically or horizontally centered, left, right, top, or bottom, and combinations of those (e.g., vertically centered and horizontally centered, horizontally centered on the bottom, etc.)

As mentioned, almost all the power and flexibility of the TWPSuperPrint component can be centrally controlled from within a dialog box-type form.

Please check out the LabelPrint demo to learn how to add simple label printing to your application.

For an example of this, see the "SuperPrint" example off of Demos\Tasks. Please feel free to use as much of that code as you wish (or the ***entire*** Print Console itself) in your own application.



The demo is written to be as generic as possible. Should you wish to customize the code, check out the assignments in FormCreate event (e.g., the EditBox and Preview properties), and tailor them to fit your particular case. Please note that the controls on the right hand side of the form could be easily copied and pasted into a different application.

To print a test booklet, simply select File / Print, everything is already set up in the demo. Out comes the pages, and correctly ordered! Optionally, if you have the product wPDF, also a PDF file can be created.

For the sake of simplicity, from here on we'll refer to the TWPSuperPrint component as WPSuperPrint1, or SuperPrint.

Booklet Printing

Let's talk about the booklet printing feature of this component.

This is driven by a single procedure, **WPSuperPrint1.SetTwoUpBooklet**(). When called with first parameter set to true, and then the printed page width and height in twips, a number of adjustments required to print in a booklet format are made behind the scenes - so that you (or the person running the application) don't have to. They include:

- setting the printer into Landscape
- controls whether the page passed to it for printing is printed on the left or right half of the page
- internally adjusts the page dimensions so that two pages can be printed on a sheet (more on this coming up)
- sets the orientation of the TWPRichText component referred to by WPSuperPrint1.EditBox to Portrait (well, more accurately, it sets Landscape to False)
- sets WPSuperPrint1's Columns property to 2, and its Rows property to 1
- sets WPSuperPrint1's Mode property to wpprPageColRows. This tells it how to calculate the page height and width
- sets all of WPSuperPrint1's margins (left, right, top, and bottom) to 0, since we already have the margins in the text

Note: be sure to set TwoUpBooklet to True ***prior*** to executing a Printer.BeginDoc statement. If you don't, SuperPrint's code that automatically changes the printer's Orientation to Landscape will have no effect.

One very important part of booklet printing is to determine the order that pages are sent to the printer. If we use as an example for this discussion a 12 page document, the pages would be ordered like this:

	Page # on	
Sheet #	left half of sheet	right half of sheet
1	12	1
2	2	11
3	10	3
4	4	9
5	8	5
6	6	7

The first sheet contains the highest and lowest numbered page. The second sheet contains the second highest and lowest numbered pages.

Notice how on the first sheet, the even numbered page is positioned on the left half and the odd numbered page on the right . However, the second sheet reverses that – the odd page

is on left, the even page on the right. WPSuperPrint1 automatically handles the left / right positioning, whichever is appropriate to the current sheet.

Your code is responsible for supplying SuperPrint with the page number to print. WPSuperPrint1's OnCalcPageNumber should point to your procedure that calculates the page number ordering. For example:

```
WPSuperPrint1.OnCalcPageNumber := DoCalcPage
```

The DoCalcPage procedure in the SuperPrint demo is a good example of how to calculate the page numbers. As written, it will work with documents any number of pages.

Another important part to using the TWPSuperPrint component is its Paint procedure. The demo's StartPrint procedure is a good example of setting up the call to WPSuperPrint1's Paint procedure, and the parameters passed to it.

A word about the call to RestoreValues in StartPrint. As mentioned above, SuperPrint internally adjusts page dimensions and orientation to print in booklet format. Therefore, you should store those properties somewhere (see the Demo's FormCreate event handler) so they can be restored after printing. And they should be saved ***prior*** to setting WPSuperPrint1's TwoUpBooklet property True. In the demo, that's what RestoreValues is about. And although the code to store the values is in FormCreate, that code could just as easily have been moved to the top of StartPrint.

Hint: By using a printer capable of supporting 11 inch x 17 inch paper, you can produce booklets with a page dimension of 8.5 inches by 11 inches (since the paper's orientation is switched to Landscape behind the scenes)!

Labels

To print labels, first, create the label (from within the TWPRichText component you assigned to the WPSuperPrint1.EditBox property), or import one into there. Then (in no particular order)

- set TwoUpBooklet to False
- include the line: WPSuperPrint1.Mode := wpprLabels
  (where WPSuperPrint1 is the TWPSuperPrint component), or add a checkbox that you use to assign/unassign the Mode property
- set Columns to the number of labels you want across the page
- set Rows to how many rows of labels you want across the page
- set Copies to the product of Rows * Columns

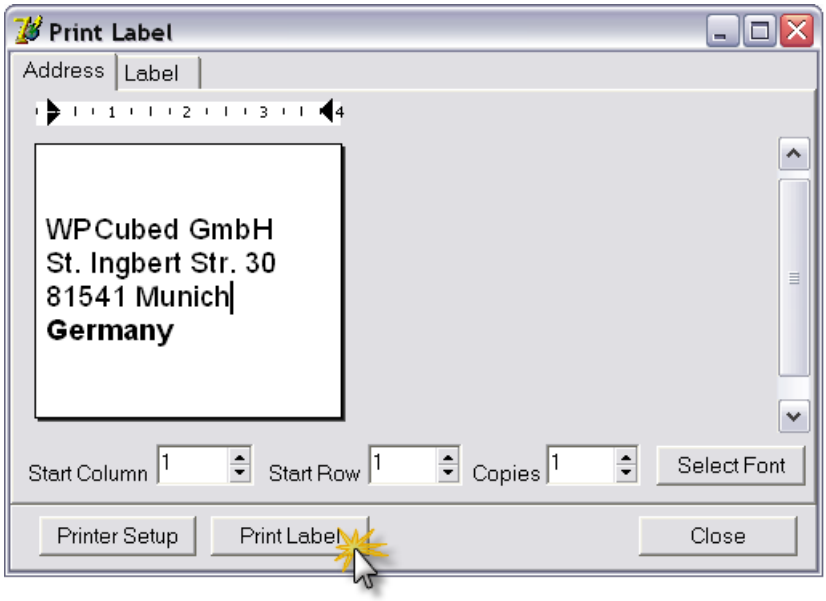
That's all you have to do to print labels!

5.3.20 Print Labels

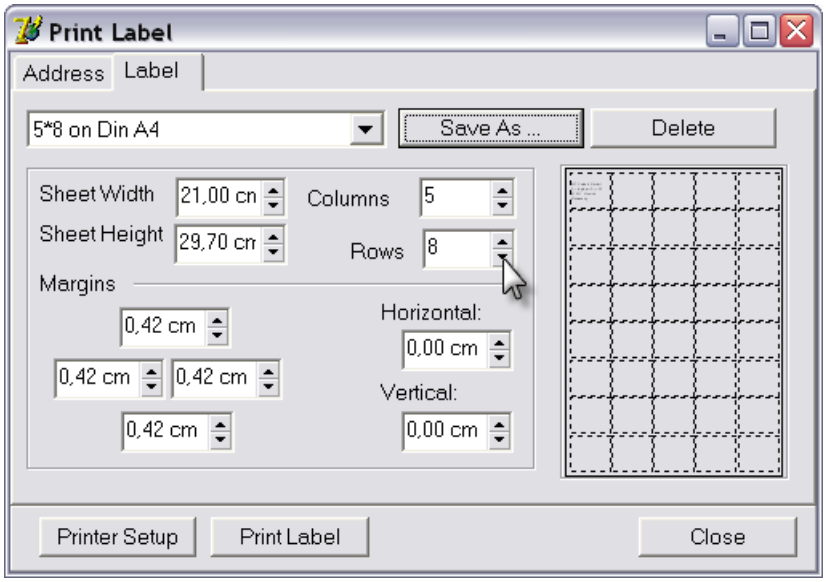
We added an easy to understand demo which shows how to use the TWPSuperPrint component to add label printing to your application.

The demo code has been created in a way which makes it easy to use it in your application.

This is a screenshot of the main dialog - Page 1:



This is a screenshot of the main dialog - Page 2:



The code also implements loading and saving of the label definitions into one XML file which will be stored in the application root path.

The format used inside this XML file is:

```
<?xml version="1.0" encoding="windows-1250"?>
<Labels>
   <Def Name="2*5_on_Din_A4">
       <TopMargin>238</TopMargin>
       <LeftMargin>238</LeftMargin>
       <RightMargin>238</RightMargin>
       <BottomMargin>238</BottomMargin>
       <HorzMargin>0</HorzMargin>
       <VertMargin>0</VertMargin>
       <ColCount>2</ColCount>
       <RowCount>5</RowCount>
       <PageWidth>11906</PageWidth>
       <PageHeight>16838</PageHeight>
   </Def>
```

To use this form you only have to create and show it.
To load the first lines of the text into the label use the procedure **LoadAddress**.

Example:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    WPLabelForm.LoadAddress( WPRichText1 );
    WPLabelForm.Show;
end;
```

On the form the user can select a font, change the text and change the label definition.

If any of values, such as the count of columns, is changed, this procedure is executed:

```
procedure TWPLabelForm.ColCountChange(Sender: TObject);
begin
  if FLocked then exit;
  WPSuperPrint1.PageWidth := PageWidth.Value;
  WPSuperPrint1.PageHeight := PageHeight.Value;

  WPSuperPrint1.Rows := RowCount.IntValue;
  WPSuperPrint1.Columns := ColCount.IntValue;
  WPSuperPrint1.MarginTop := TopMargin.Value;
  WPSuperPrint1.MarginLeft := LeftMargin.Value;
  WPSuperPrint1.MarginRight := RightMargin.Value;
  WPSuperPrint1.MarginBottom := BottomMargin.Value;
  WPSuperPrint1.InbetweenHorz := HorzMargin.Value;
  WPSuperPrint1.InbetweenVert := VertMargin.Value;

  WPSuperPrint1.LabelStartRow := StartRow.Value;
  WPSuperPrint1.LabelStartColumn := StartCol.Value;
  WPSuperPrint1.Copies := Copies.Value;

  WPRichText1.Header.SetPageWH(
    WPSuperPrint1.Width,
    WPSuperPrint1.Height,
    0, 0, 0, 0);
  WPRichText1.ReformatAll;
end;
```

It sets the properties of the SuperPrint component. After that the properties Width and Height which reflect the current label size are applied to the TWPRichText - it holds the text of the label and is used for the printing process.

The printing is started with this code:

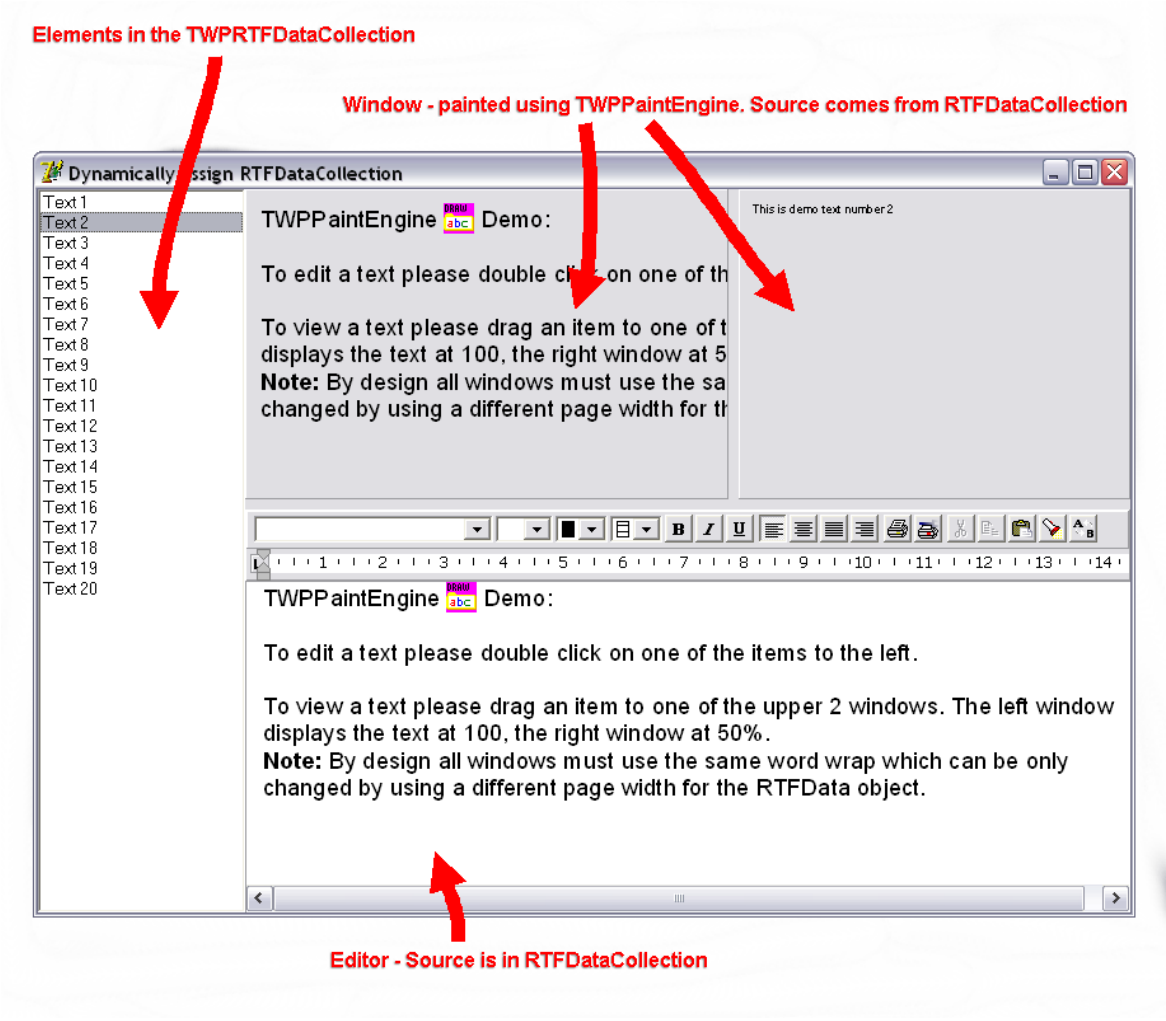
```
procedure TWPLabelForm.Button1Click(Sender: TObject);
begin
  Printer.Title := 'Label';
  Printer.BeginDoc;
  WPSuperPrint1.Paint(
```

```
Printer.Canvas,
    -GetDeviceCaps(Printer.Handle,  PHYSICALOFFSETX), // Offset in pixels
    -GetDeviceCaps(Printer.Handle,  PHYSICALOFFSETy), // Offset in pixels
    GetDeviceCaps(Printer.Handle,  LOGPIXELSY) / 1440, // Multiplicator  for Parameters
(twips->Canvas)
    [wpDoNotScalePage]); // Options
  Printer.EndDoc;
end;
```

It uses the Paint procedure of the SuperPrint component to render one page to the printer canvas. The physical offsets are passed as negative values to make sure the positions are acurate. Of course this offsets can be used to adjust the printing.

5.3.21 Print/Edit elements of TWPRTFDataCollection

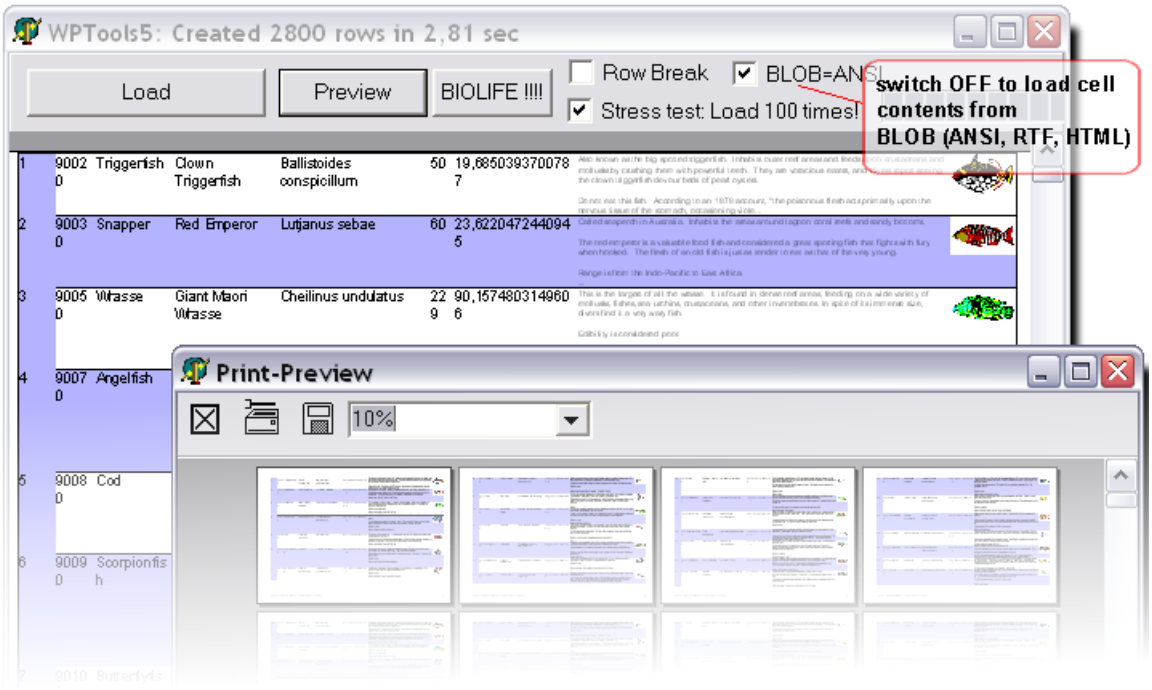
The demo **DynAssignRTFData** shows how to use the TWPPaintEngine to paint elements stored in a TWPRTFDataCollection on any Canvas. It also shows how to dynamically assign the RTFData object to an editor - so the editor can edit any of the elements in the collection!



5.3.22 Create Table from Database

WPTools possibilities to create text and tables in code are extraordinary powerful.

We created an example (Demos\GridMode) which loads a BDE database and creates a table with all the rows. Optionally blobs can be loaded using the TParagraph.LoadFromStream method - this will preserve the formatting.
The demo also shows how to change the background color using the ASetColor procedure. Optionally rows can be splitted on several pages (RowBreak) and, if you want to test the performance, the same data can be imported 100 times.



This screen shot was taken after the well known BIOLIFE database was loaded - 100 times. This created 2800 rows and images on 400 pages in only 2.8 seconds! (P-M 1.6 GHZ)

This is the main routine of the demo.
It first cleares the text and sets the page size. Then a footer is created to show the page number. After that a new table paragraph is created (table) which will then host all rows.

Please note that the *CreateRow* function creates an object of the class TWPTableRowStyle. This class inherits from TWPTextStyle (so does class TParagraph!) and is used as template for all cells which are created. So when you make changes to rowstyle, the new properties will be applied to all cells which are created afterwards using *InputCell*.
The rowstyle object is deleted when the row is completed with *EndRow*.

```
procedure TForm1.LoadFromDataSet(Data: TDataSet; aName: string; LoadBlobAsANSI: Boolean);
var i, a, a_max, RowNr: Integer;
  table, cell, row: TParagraph;
  b: Boolean;
  obj: TWPTextObj;
  wpobj: TWPObject;
  bit: TBitmap;
  rowstyle: TWPTableRowStyle;
  tim: Cardinal;
  stream: TStream;
```

```
begin
  WPRichText1.Clear;
  Caption := 'loading... - press ESCAPE to abort';
  tim := GetTickCount;
  try

    WPRichText1.EditOptions := [];

  // Set Page Size
    WPRichText1.Header.PageSize := wp_DinA4;
    WPRichText1.Header.LeftMargin := WPCentimeterToTwips(2);
    WPRichText1.Header.RightMargin := WPCentimeterToTwips(1);
    WPRichText1.Header.TopMargin := WPCentimeterToTwips(1.5);
    WPRichText1.Header.BottomMargin := WPCentimeterToTwips(1.5);
    WPRichText1.Header.Landscape := TRUE;
    // WPRichText1.WordWrap := TRUE;

    // Create Footer
    WPRichText1.ActiveText := WPRichText1.HeaderFooter.Get(
           wpIsFooter, wpraOnAllPages);
    WPRichText1.InputString(aName + #9);
    WPRichText1.InputTextField(wpoPageNumber);
    WPRichText1.ASet(WPAT_BorderFlags, WPBRD_DRAW_Top);
    WPRichText1.SetTabPos(MaxInt, tkRight);

     // Switch to BODY
    WPRichText1.ActiveText := WPRichText1.BodyText;

    RowNr := 0;

    if StressTest.Checked then
    begin a_max := 100;
      ProgressBar1.Visible := TRUE;
    end else
    begin
      a_max := 1;
      ProgressBar1.Visible := FALSE;
    end;

    // Boolean to alternate the background
    b := FALSE;

    // Add all rows to this table
    table := WPRichText1.ActiveText.CreateTable(nil);

    table.ASet(WPAT_BorderFlags, WPBRD_DRAW_All4);

    // now create the rows, a_max is used for stresstest
    for a := 1 to a_max do
    begin
      ProgressBar1.Position := a;
      ProgressBar1.Update;

    // Start at the beginning of database
      Data.First;

    // Repeat for all data rows
      repeat
        inc(RowNr);
        rowstyle := table.CreateRow(nil, true);
        if rowstyle <> nil then
        begin
          b := not b;
          rowstyle.ASetColor(WPAT_BGColor, clBlue);
          rowstyle.ASet(WPAT_ShadingValue, 30);

        // Create first Column with numbers
          cell := rowstyle.InputCell;
          cell.ASet(WPAT_BorderFlags, WPBRD_DRAW_Right);
          cell.ASet(WPAT_COLWIDTH, WPCentimeterToTwips(1));
          cell.SetText(IntToStr(RowNr));

        // Make sure every other row is *not* shaded:
```

```
if b then
          begin
            rowstyle.ADel(WPAT_BGColor);
            rowstyle.ADel(WPAT_ShadingValue);
          end;
          rowstyle.ASet(WPAT_IndentRight, 72);

          for i := 0 to Data.Fields.Count - 1 do
          begin
            cell := rowstyle.InputCell;
            if Data.Fields[i] is TGraphicField then
            begin
              bit := TBitmap.Create;
              bit.Assign(Data.Fields[i]);
              wpobj := TWPOImage.CreateImage(WPRichText1.Memo.RTFData, bit);
              obj := TWPTextObj.Create;
              cell.Insert(0, obj);
              obj.ObjRef := wpobj;
              obj._SetObjType(12); // = TWPTextObjType.wpobjImage
              obj.Width := wpobj.ContentsWidth div 3;
              obj.Height := wpobj.ContentsHeight div 3;
              bit.Free;
            end
            else if Data.Fields[i] is TBlobField then
            begin
              if LoadBlobAsANSI then
              begin
            // The simple method which loads text into one paragraph
                cell.ASet(WPAT_CharFontSize, 600);
                cell.SetText(Copy(Data.Fields[i].AsString, 1, 400) + '...');
              end else
              begin
            // the "difficult" method which also loads formatted text
                stream := TBlobStream.Create(Data.Fields[i] as TBlobField, bmRead);
                try
                  cell.LoadFromStream(stream,
                    'AUTO', '', [wploadpar_ClearShading]);
                finally
                  stream.Free;
                end;
              end;
            end
            else cell.SetText(Data.Fields[i].AsString);
            cell.ASet(WPAT_BorderFlags, WPBRD_DRAW_Bottom);
          end;
        // Create the cells
          row := table.EndRow(rowstyle);
          if not RowBreak.Checked then
            row.ASet(WPAT_ParKeep, 1);

        // Allow ESCAPE
          if (GetAsyncKeyState(VK_ESCAPE) shr 15) <> 0 then
          begin
            if MessageBox(Handle, 'Abort loading of data ?', 'ESCAPE',
              MB_YESNO) = IDYES then exit;
          end;
        end;

        Data.Next;
      until Data.EOF;
    end; // for a

  finally
    WPRichText1.Refresh;
    Caption := Format('WPTools5: Created %d rows in %.02f sec', [RowNr, (GetTickCount -
tim) / 1000]);
  end;
end;
```

5.3.23 Print Watermarks

Watermarks are drawings which are printed before the actual contents of a page is printed.

In WPTools 5 Watermarks have to be printed inside the event handler for **OnPaintWatermark**.

This event gets this parameters:

Sender a reference to the TWPRichText component which tiggered the event
RTFEngine is the RTF Engine (= TWPRichText.Memo)
toCanvas the drawing canvas for the output. During printtime this is the printer canvas.
PageRect the bounding rectangle of the page. During printime the Top,Left coordinate is set to the negative physical offset to make it easy to print at exact positions.
PaintPageNr the number of the page, 0 based
RTFPageNr 0 or the number of the text page which is printed later over the watermark. You can access the definition object (class TWPVirtPage) of this page by reading `RTFEngine.DisplayedText.Pages[RTFPageNr - 1]`.
WaterMarkRef reserved
XRes and **YRes** define the current resolution. This is very important to convert real coordinates into coordinates on the virtual paper. You can use this functions to convert CM into pixles. The result value has to be added to PageRect.Left or PageRect.Top.

```
function XP(cm: Double): Integer;
begin
  Result := MulDiv(WPCentimeterToTwips(cm), Xres, 1440);
end;
function YP(cm: Double): Integer;
begin
  Result := MulDiv(WPCentimeterToTwips(cm), Yres, 1440);
end;
```

CurrentZoom specifies the zooming the editor uses, at printtime it is 1
PaintMode the paint mode which is currently used. By checking for 'wppInPaintDesktop' you can write code which is not active at print time.

Examples how Watermarks can be used (Project: Demos\Tasks\WaterM2) **:**

a) show a background pattern on the virtual paper - but do not print this pattern:

```
var x,y : Integer;
    bit : TBitmap;
begin
  if wppInPaintDesktop in PaintMode then
  begin
    x := PageRect.Left;
    bit := Image1.Picture.Bitmap;
    while x<PageRect.Right do
    begin
     y := PageRect.Top;
     while y<PageRect.Bottom do
     begin
       toCanvas.Draw(x,y,bit);
       inc(y, bit.Height);
     end;
     inc(x, bit.Width);
    end;
  end;
end;
```

b) Draw a 0.5 cm grid using light blue color, this grid is also printed

```
var i,j : Integer;
begin
    toCanvas.Pen.Width := 0;
    toCanvas.Pen.Color := $00FAD5AF;
    toCanvas.Pen.Style := psSolid;
    for i:=1 to 1000 do
    begin
      j := PageRect.Left + MulDiv(WPCentimeterToTwips(0.5 * i), Xres, 1440);
      if j>= PageRect.Right then break;
      toCanvas.MoveTo(j, PageRect.Top);
      toCanvas.LineTo(j, PageRect.Bottom);
    end;

    for i:=1 to 1000 do
    begin
      j := PageRect.Top + MulDiv(WPCentimeterToTwips(0.5 * i), Yres, 1440);
      if j>= PageRect.Bottom then break;
      toCanvas.MoveTo(PageRect.Left, j);
      toCanvas.LineTo(PageRect.Right, j);
    end;
end;
```

c) Print a frame line around the text area

(1) We are using the main page layout information:

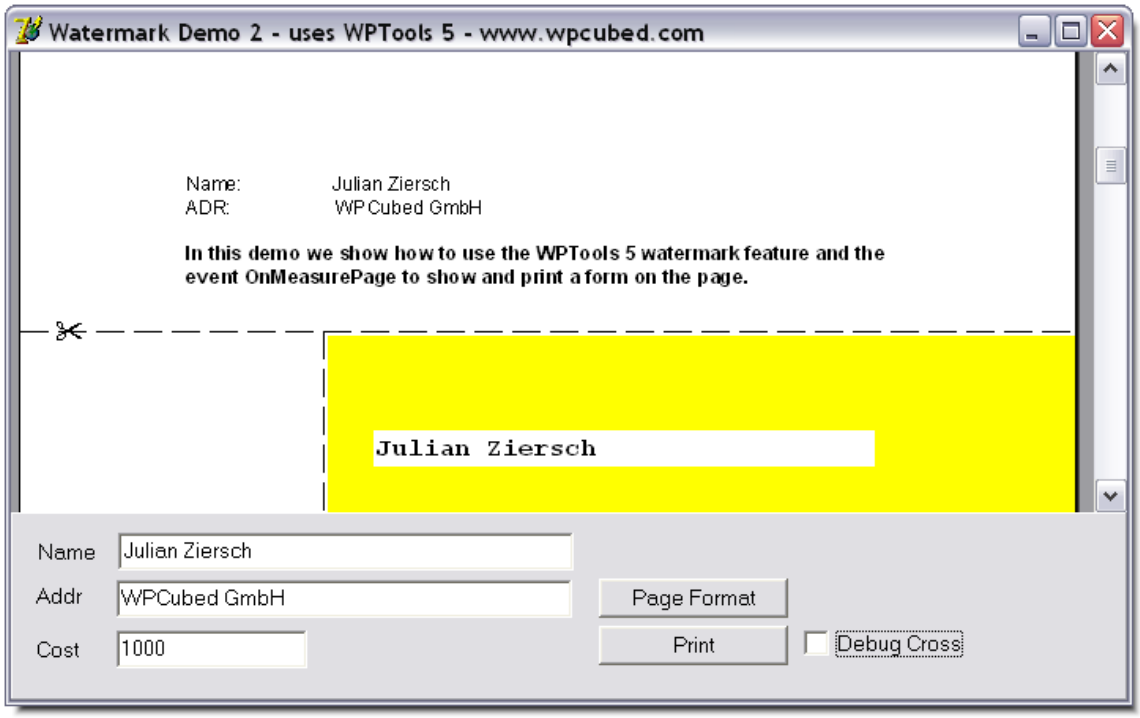
```
procedure TWaterM.WPRichText1PaintWatermark(Sender: TObject;
   RTFEngine: TWPRTFEnginePaint; toCanvas: TCanvas;
   PageRect: TRect; PaintPageNr, RTFPageNr: Integer; WaterMarkRef: TObject;
   XRes, YRes: Integer;
   FCurrentZoom : Single;
   PaintMode: TWPPaintModes);
var r: TRect;
begin
  r.Left := PageRect.Left + MulDiv( RTFEngine.RTFData.Header.LeftMargin, XRes, 1440);
  r.Top := PageRect.Top + MulDiv( RTFEngine.RTFData.Header.TopMargin, YRes, 1440);
  r.Right := PageRect.Right - MulDiv( RTFEngine.RTFData.Header.RightMargin, XRes, 1440);
  r.Bottom := PageRect.Bottom - MulDiv( RTFEngine.RTFData.Header.BottomMargin, YRes,
1440);
  toCanvas.Pen.Color := clBtnShadow;
  toCanvas.Pen.Width := 0;
  toCanvas.Brush.Style := bsClear;
  toCanvas.Rectangle(r);
end;
```

(2) Now we are using the page layout information for the current page.

```
  if RTFPageNr > 0 then
  begin
    r.Left := PageRect.Left + MulDiv(
         RTFEngine.DisplayedText.Pages[RTFPageNr - 1].PageMarginLeft, XRes, 1440);
    r.Top := PageRect.Top + MulDiv(
         RTFEngine.DisplayedText.Pages[RTFPageNr - 1].PageMarginTop, YRes, 1440);
    r.Right := PageRect.Right - MulDiv(
         RTFEngine.DisplayedText.Pages[RTFPageNr - 1].PageMarginRight, XRes, 1440);
    r.Bottom := PageRect.Bottom - MulDiv(
         RTFEngine.DisplayedText.Pages[RTFPageNr - 1].PageMarginBottom, YRes, 1440);
    toCanvas.Pen.Color := clBtnShadow;
    toCanvas.Pen.Width := 0;
    toCanvas.Brush.Style := bsClear;
    toCanvas.Rectangle(r);
  end;
```

Please note that the parameter *RTFPageNr* can be 0 if the page was inserted as "external page".

d) Display a pre-printed form (such as a money transfer form) as part of the page.



This example requires to use the event OnMeasurePage as well. In this event we reserve space at the bottom of the first page:

```
const PR_Form_Height = 10.5; // Form Height in CM
      PR_Form_Width  = 15.0;

procedure TForm1.WPRichText1MeasureTextPage(Sender: TObject;
  PageInfo: TWPMeasurePageParam);
begin
  // We want to make sure the first page has a bottom margin which is
  // large enough for our form
  if PageInfo.pagenr = 1 then
  begin
    PageInfo.marginbottom := WPCentimeterToTwips(PR_Form_Height);
    PageInfo.changed := TRUE;
  end;
end;
```

The PaintWatermark code only draws on the first page:

```
procedure TForm1.WPRichText1PaintWatermark(Sender: TObject;
  RTFEngine: TWPRTFEnginePaint; toCanvas: TCanvas; PageRect: TRect;
  PaintPageNr, RTFPageNr: Integer; WaterMarkRef: TObject; XRes,
  YRes: Integer; CurrentZoom: Single; PaintMode: TWPPaintModes);
  // ~~~~~~~~~~~~~~~~~~~~ Convert CM values into pixel ~~~~~~~~~~~~~
  function XP(cm: Double): Integer;
  begin
    Result := MulDiv(WPCentimeterToTwips(cm), Xres, 1440);
  end;
  function YP(cm: Double): Integer;
  begin
    Result := MulDiv(WPCentimeterToTwips(cm), Yres, 1440);
  end;
  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
var r, r2: TRect;
  off, w: Integer;
begin
  if PaintPageNr = 0 then
```

```
begin
    r := PageRect;
    r.Top := r.Bottom - YP(PR_Form_Height);
    toCanvas.Pen.Color := clBlack;
    toCanvas.Pen.Style := psDash;
    toCanvas.MoveTo(r.Left, r.Top);
    toCanvas.LineTo(r.Right, r.Top);

    // Draw the form at the right bottom border
    r.Left := r.Right - XP(PR_Form_Width);

    // Draw line
    toCanvas.MoveTo(r.Left, r.Top);
    toCanvas.LineTo(r.Left, r.Bottom);

    // Draw the sizzors
    toCanvas.Font.Name := 'WingDings';
    toCanvas.Font.Height := -YP(0.5);
    toCanvas.TextOut( PageRect.Left + XP(0.7), r.Top-
       toCanvas.TextHeight(#$22) div 2 , #$22 );

     // This is background of the form, we do not draw this when
     // we are printing!
    if wppInPaintDesktop in PaintMode then
    begin
      r2 := r;
      inc(r2.Left,XP(0.1));
      inc(r2.Top,YP(0.1));
      toCanvas.Brush.Color := clYellow;
      toCanvas.FillRect(r2);
    end;

     // This are the form text
    toCanvas.Brush.Color := clWhite;
    toCanvas.Font.Name := 'Courier New';
    toCanvas.Font.Height := -YP(0.5);
    toCanvas.Font.Style := [fsBold];
    off := YP(0.1);

     // NAME
    r2.Left := r.Left + XP(1.0);
    r2.Top := r.Top + YP(2);
    r2.Right := r2.Left + XP(10);
    r2.Bottom := r2.Top + YP(0.7);
    toCanvas.FillRect(r2);
    toCanvas.TextOut(r2.Left + off, r2.Top + off, NameE.Text);

     // ADR
    r2.Left := r.Left + XP(1.0);
    r2.Top := r.Top + YP(5.5);
    r2.Right := r2.Left + XP(10);
    r2.Bottom := r2.Top + YP(0.7);
    toCanvas.FillRect(r2);
    toCanvas.TextOut(r2.Left + off, r2.Top + off, AdrE.Text);

     // COST, right aligned
    r2.Left := r.Left + XP(8.5);
    r2.Top := r.Top + YP(4.5);
    r2.Right := r2.Left + XP(5);
    r2.Bottom := r2.Top + YP(0.7);
    toCanvas.FillRect(r2);
    w := toCanvas.TextWidth(CostE.Text);
    toCanvas.TextOut(r2.Right - off - w, r2.Top + off, CostE.Text);

    // For Debugging
    // Draw a Line at 10,10 CM
    if DrawDebugCross.Checked then
    begin
      toCanvas.TextOut( PageRect.Left+XP(10), PageRect.Top, '10');
      toCanvas.TextOut( PageRect.Left, PageRect.Top+YP(10), '10');
      toCanvas.MoveTo( PageRect.Left, PageRect.Top + YP(10));
      toCanvas.LineTo( PageRect.Right, PageRect.Top + YP(10));
      toCanvas.MoveTo( PageRect.Left+XP(10), PageRect.Top);
```

```
toCanvas.LineTo( PageRect.Left+XP(10), r.Top);
    end;
  end;
end;
```

5.3.24 Use "External Pages"

What are "external pages" ?

This are pages which are displayed by WPTools 5 before, after or within the regular text. In page layout mode the page will be displayed as if it was a text page.

Why do I need it ?

If you want to display the output of your favourite report generator and text in a powerful preview component. For example if you need to create a big report which contains of several parts, text and graphical report data.
This feature will also help a lot if you need to combine the ANSI text (such as logging data) with formatted text. Since the external pages are rendered through an event it is possible to work with thousands of them - there is no need to initialize the graphical data at the beginning.

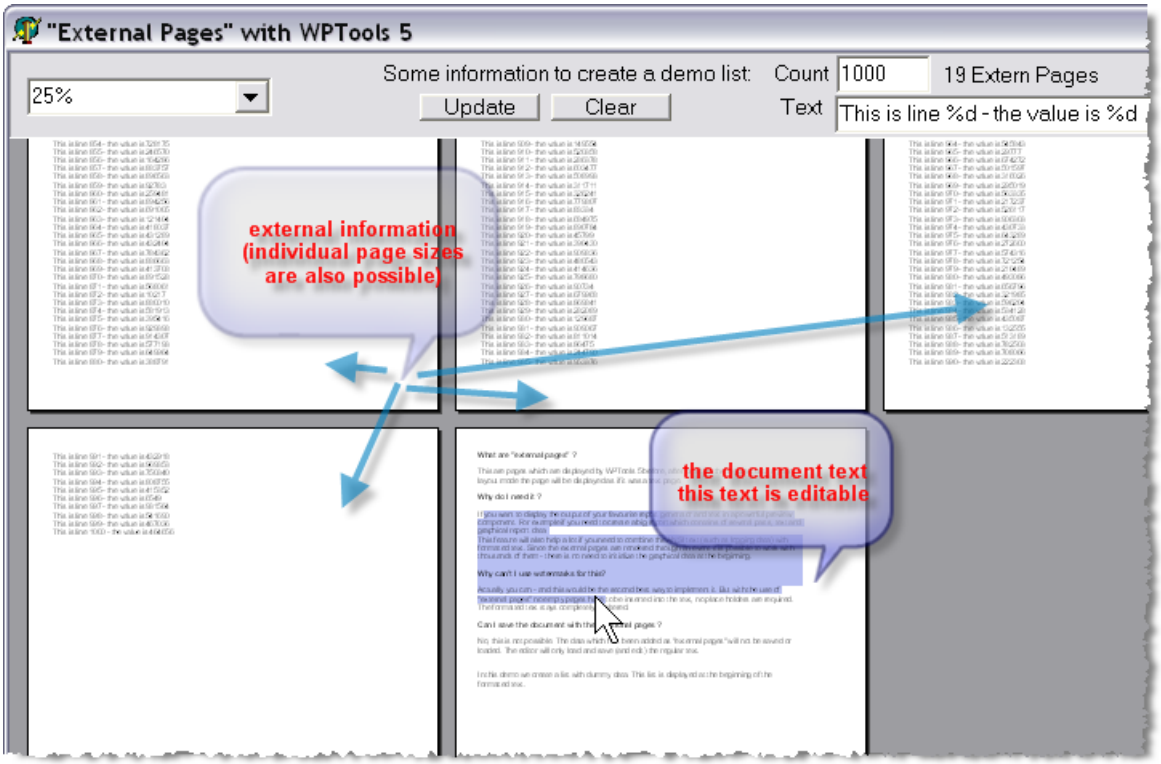
Why can't I use watermarks for this?

Actually you can - and this would be the second best way to implement it. But with the use of "external pages" no empty pages have to be inserted into the text, no place holders are required. The formatted text stays completely unaltered.

Can I save the document with the additional pages ?

No, this is not possible. The data which has been added as "external pages" will not be saved or loaded. The editor will only load and save (and edit) the regular text.

The installed (Tasks\ExternalPages) demo creates a list with dummy data. This list is displayed at the beginning of the formatted text:



Methods to be used for **external pages:** (only in TWPRTFEnginePaint = WPRichText.Memo)

Adds a reference 'ExternPageRef'. The returned TWPRTFExternPageRef contains variables to change the page size.

function ExternPageRefAdd(
  mode: TWPRTFExternPageRefMode;
  PageNrX: Integer;
  Order: Integer;
  ExternPageRef: TObject;
  WatermarkRef: TObject): TWPRTFExternPageRef;

procedure ExternPageRefClear - deletes all references and frees the data

Events to be used with external pages:

OnPaintExternPage (in TWPCustomRtfEdit and TWPRTFEnginePaint)

```
Sender: TObject;
RTFEngine: TWPRTFEnginePaint;
prCanvas: TCanvas;
xoff, yoff: Integer;
r: TRect;
PaintPageNr: Integer;
ExternPageRef: TObject;
DestXRes, DestYRes: Integer
```

OnInitPage (only in TWPRTFEnginePaint = WPRichText.Memo)

```
Sender: TObject; // The 'Parent' Object of the RTF - Engine, usually the TWPRichText
RTFEngine: TWPRTFEnginePaint;
```

```
paintpagenr: Integer; // the number of the paint page (this is not the RTF page)
    rtfpagenr: Integer; // the RTF page which will be used if ExternPageRef stayes to be
nil
    var ExternPageRef: TObject; // assign an object here to disable RTF text for this page
    var WatermarkRef: TObject; // assign data required to paint a watermark
    var PaperColor: TColor;
    var pagewidth, pageheight, marginleft, margintop, marginright, marginbottom: Integer)
of object;
```

Example:

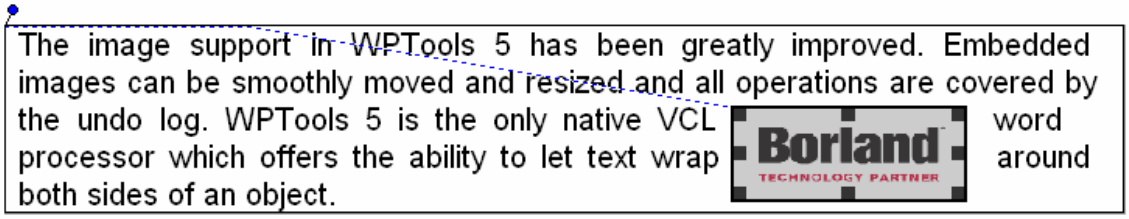
```
procedure TWPExternalP.UpdateBtnClick(Sender: TObject);
var s       : string;
    c,i     : Integer;
    x,y,th,h: Integer;
    pw, ph  : Integer;
    PageNr  : Integer;
    aPage   : TMetafile;
    aCanvas : TMetafileCanvas;
    procedure NewPage;
    begin
      if aPage = nil then
      begin
         aPage := TMetafile.Create;
         aPage.Width := pw;
         aPage.Height := ph;
      end;
      if aCanvas=nil then
      begin
        aCanvas := TMetafileCanvas.Create( aPage, 0);
        aCanvas.Font.Name := 'Arial';
        aCanvas.Font.Size := 11;
        h := ph - WPScreenPixelsPerInch;
        y := WPScreenPixelsPerInch div 2;
        th:= aCanvas.TextHeight('Ag');
        x := WPScreenPixelsPerInch div 2;
      end;

      PageCount.Caption := IntToStr(PageNr+1) + ' Extern Pages';
      PageCount.Update;
    end;
    procedure PostPage;
    var PageRef : TWPRTFExternPageRef;
    begin
      FreeAndNil(aCanvas);
      if aPage<>nil then
      begin
         // We add a page at a certain location. The data object we added
         // Is freed automatically unless we set in the
         // returned TWPRTFExternPageRef object the property DontFreeExternPage
         // to true
         PageRef := WPRichText1.Memo.ExternPageRefAdd(
             wpAsPageX,
             PageNr,
             0,
             aPage,
             nil
            );
         // This values are optional
         PageRef.PageWidth  := MulDiv( pw, 1440, WPScreenPixelsPerInch);
         PageRef.PageHeight := MulDiv( ph, 1440, WPScreenPixelsPerInch);

        inc(PageNr);
        aPage := nil;
      end;
    end;
begin
   c := StrToInt(ACount.Text);  // The count of lines to be printed
   s := ALine.Text; // the format string
   aPage    := nil;
```

```
aCanvas  := nil;
   PageNr   := 0;
   WPRichText1.Memo.ExternPageRefClear;

   // The page size for the external page in pixels
   pw := MulDiv( WPRichText1.Header.PageWidth, WPScreenPixelsPerInch, 1440);
   ph := MulDiv( WPRichText1.Header.PageHeight, WPScreenPixelsPerInch, 1440);

   // Try it: The external pages will be landscape!
   // ph := MulDiv( WPRichText1.Header.PageWidth, WPScreenPixelsPerInch, 1440);
   // pw := MulDiv( WPRichText1.Header.PageHeight, WPScreenPixelsPerInch, 1440);


   // Now create the lines
   for i:=1 to c do
   begin
      if y>h then PostPage;
      NewPage;
      aCanvas.TextOut(x,y,Format(s,[i,Random( 1000000)]));
      inc(y,th);
   end;
   PostPage;

   WPRichText1.DelayedReformat;
end;
```

Since we use simple metafiles as data for the external pages the painting is very easy. Of course we could also use a string list or a custom object and so avoid the overhead of metafiles. If you know in advance the count of pages but do not want to load the data in the beginning, this is possible, too. The data can be initializes when it is used first in the event OnPaintExternPage.

```
procedure TWPExternalP.WPRichText1PaintExternPage(Sender: TObject;
  RTFEngine: TWPRTFEnginePaint; prCanvas: TCanvas; xoff, yoff: Integer;
  r: TRect; PaintPageNr: Integer; ExternPageRef: TObject; DestXRes,
  DestYRes: Integer);
begin
  prCanvas.StretchDraw( r, ExternPageRef as TGraphic );
end;
```

5.3.25 Images



(This screenshot shows text and a floating object in WPTools 5.)

Provide a Graphic Popup Menu:

Please insert this popup menu ito your form:

```
object GraphicPopupMenu: TPopupMenu
  Left = 675
  Top = 443
  object ascharacter1: TMenuItem
    Tag = 1
    Caption = 'as character'
  end
```

```
object reltoparautowrap1: TMenuItem
    Tag = 2
    Caption = 'rel. to par - auto wrap left or right'
  end
  object reltoparwrapleftandright1: TMenuItem
    Tag = 3
    Caption = 'rel to par - wrap left and right'
  end
  object reltopagenowrappng1: TMenuItem
    Tag = 4
    Caption = 'rel. to page - no wrappng'
  end
  object reltopagewrapleftandright1: TMenuItem
    Tag = 5
    Caption = 'rel. to page - wrap left and right'
  end
end
```

Now select all items and create a one OnClick event for all menu items:

In **C++ Builder** use this code:

```
void __fastcall TForm1::GraphicOptionsClick(TObject *Sender)
{
  if (WPRichText1->SelectedObject)
  {
    switch ((((TComponent *)Sender)->Tag))
    {
      case 1: WPRichText1->SelectedObject->PositionMode = wpotChar;
         break;
      case 2:
          WPRichText1->SelectedObject->Wrap = wpwrAutomatic;
          WPRichText1->SelectedObject->PositionMode = wpotPar;
        break;
      case 3:
          WPRichText1->SelectedObject->Wrap = wpwrBoth;
          WPRichText1->SelectedObject->PositionMode = wpotPar;
        break;
      case 4:
          WPRichText1->SelectedObject->Wrap = wpwrNone;
          WPRichText1->SelectedObject->PositionMode = wpotPage;
        break;
      case 5:
          WPRichText1->SelectedObject->Wrap = wpwrBoth;
          WPRichText1->SelectedObject->PositionMode = wpotPage;
        break;
     }
   }
}
```

In **Delphi** You can use this code:

```
procedure TWPForm1.GraphicOptionsClick(Sender: TObject);
begin
  if (WPRichText1<>nil) and (WPRichText1.SelectedObject <> nil) then
    case (Sender as TComponent).Tag of
      1: WPRichText1.SelectedObject.PositionMode := wpotChar;
      2:
        begin
          WPRichText1.SelectedObject.Wrap := wpwrAutomatic;
          WPRichText1.SelectedObject.PositionMode := wpotPar;
        end;
      3:
        begin
          WPRichText1.SelectedObject.Wrap := wpwrBoth;
          WPRichText1.SelectedObject.PositionMode := wpotPar;
        end;
      4:
        begin
          WPRichText1.SelectedObject.Wrap := wpwrNone;
```

```
WPRichText1.SelectedObject.PositionMode  := wpotPage;
        end;
      5:
        begin
          WPRichText1.SelectedObject.Wrap  := wpwrBoth;
          WPRichText1.SelectedObject.PositionMode  := wpotPage;
        end;
    end;
end;
```

Insert a graphic:

```
procedure TForm1.InsertGraphicClick(Sender: TObject);
var txtobj: TWPTextObj;
begin
  if OpenDialog2.Execute and (WPRichText1.ActiveParagraph <> nil) then
  begin
    WPRichText1.SetFocus;
    txtobj := WPRichText1.Memo.RTFData.TextObjects.Insert(
      WPLoadObjectFromFile(
          WPRichText1.Memo.RTFData,
          OpenDialog2.FileName), 1440, 1440);
   // Code to change the graphic, for example change width and height ot the
'PositionMode'
    if txtobj <> nil then
    begin


    end;
    WPRichText1.Refresh;
  end;
end;
```

Modify selected object:

```
procedure TForm1.ChangeObjectPositionAndWrapMode(Sender: TObject);
begin
   if WPRichText1.TextObjects.SelectedObj <>nil then
   begin
        WPRichText1.TextObjects.ChangePositionMode(
             WPRichText1.TextObjects.SelectedObj,
        wpotPar, wpwrBoth);
   end;
end;
```

Load new image into object:

```
procedure TForm1.Loadimage1Click(Sender: TObject);
begin
  if (WPRichText1.TextObjects.SelectedObj<>nil) and
      (OpenPictureDialog1.Execute) then
      begin
         WPRichText1.TextObjects.SelectedObj.LoadObjFromFile(
            OpenPictureDialog1.FileName);
      end;
end;
```

Replace text with a graphic file

```
var GSFile: string;
    TextObject: TWPOImage; // from unit WPObj_Image
begin
  GSFile := 'C:\testfile.jpg';
  if FileExists(GSFile) then
    with WPRichText1 do
    begin
      Finder.ToStart;
```

```
while Finder.Next('<<Graphic-Signature>>' ) do
      begin
        TextObject := TWPOImage.Create(WPRichText1.Memo.RTFData); // !
        TextObject.LoadFromFile(GSFile);
        SetSelPosLen(Finder.FoundPosition,  Finder.FoundLength);
        TextObjects.Insert(TextObject);
      end;
    end;
end;
```

Hint: It would be also possible to use modify WPRichText1.TextObjects.SelectedObj.Wrap and WPRichText1.TextObjects.SelectedObj.PositionMode directly but this change is not logged for undo.

Technical Information:

In previous versions of WPTools embedded objects were stored by using a list of references. A tag, which was required to identify a reference, was stored in the TAttr record which was used parallel to the character.

WPTools 5 uses a similar method which is much more powerful and consumes less memory.

Each paragraph can have a CharObjectIndex array. If no objects are used, this array is not allocated. The items in this array are the index values +1 in the FWPTextObjs array of the same paragraph.

If a character is an embedded object the entry in the corresponding CharObjectIndex item is set to value <> 0. (Using this double reference makes it quicker to test whether a character is an object or not.)

To check if a character is an embedded object, please use the IsObject function of the TParagraph object.

5.3.26 TWPTextObj with custom draw event

Instances of the TWPTextObj class with ObjType set to wpobjTextObj are used to display page numbers, time and similar fields. Unlike mail merge fields those objects are represented using just one character. So it is impossible to have a line wrap in the text.

Normally the objects are painted using an internal routine, so for objects with the name 'PAGE' the current page number is inserted.

The text objects are created using
WPRichText1.**InputTextFieldName**('CHANGEME');

To have a different text you can provide an event handler for the **OnTextObjGetTextEx** event:

```
procedure TForm1.WPRichText1TextObjGetTextEx(RefCanvas: TCanvas;
  TXTObject: TWPTextObj; var PrintString: WideString; var WidthInPix,
  HeightInPix: Integer; var PaintObject: TWPTextObj; Xres, YRes: Integer);
begin
  if TXTObject.Name='CHANGEME' then
  begin
      PrintString := 'more...';
  end;
```

```
end;
```

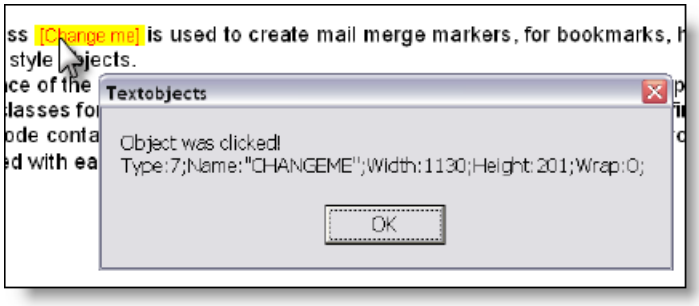
This will display the text object with the text "more..." using the attributes defined for that object.

It is very easy to create a **different background color**:

```
procedure TForm1.WPRichText1TextObjGetTextEx(RefCanvas: TCanvas;
  TXTObject: TWPTextObj; var PrintString: WideString; var WidthInPix,
  HeightInPix: Integer; var PaintObject: TWPTextObj; Xres, YRes: Integer);
begin
  if TXTObject.Name='CHANGEME' then
  begin
      PrintString := 'more...';
      RefCanvas.Brush.Color := clYellow;
  end;
end;
```

Now, if you want to **make that object clickable** you can add an event handler to the OnTextObjectClick event:

```
procedure TForm1.WPRichText1TextObjectClick(Sender: TWPCustomRtfEdit;
  pobj: TWPTextObj; obj: TWPObject; var ignore: Boolean);
begin
  if (pobj.ObjType=wpobjTextObject) and
     (pobj.Name='CHANGEME') then
  begin
     // locate next data record or similar ...
     ShowMessage('Object was clicked!' + #13 + pobj.AGetWPSS );
  end;
end;
```



Usually text objects can not be selected like images. This feature can be activated in EditOptionsEx, flag wpTextObjectSelecting. If you need to avoid the selection for certain objects add an event handler for the event BeforeObjectSelection.

```
procedure TForm1.WPRichText1BeforeObjectSelection(Sender: TObject;
  txtobj: TWPTextObj; var Ignore: Boolean);
begin
   // We only want images to be selectable
   Ignore := txtobj.ObjType <> wpobjImage;
end;
```

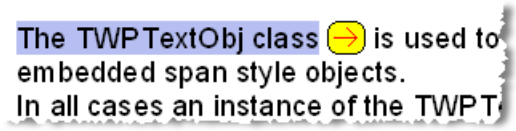
But what if you want to display some kind of graphic instead of the text?
Do not use the OnTextObjectPaint event for this - this event is used to draw embedded images only. Instead create an event handler for the objects own paint event:

```
procedure TForm1.OnPaintMarker(Sender : TWPTextObj;
    OutCanvas  : TCanvas; XRes, YRes : Integer; X, Y, W, H, BASE: Integer );
```

Assign the address of this paint event to the property **TXTObject.OnPaint** in the event OnTextObjGetTextEx and don't forget to also specify a width. The width is only used if the variable PaintObject has been set since otherwise always the PrintString is evaluated.

```
procedure TForm1.WPRichText1TextObjGetTextEx(RefCanvas: TCanvas;
  TXTObject: TWPTextObj; var PrintString: WideString; var WidthInPix,
  HeightInPix: Integer; var PaintObject: TWPTextObj; Xres, YRes: Integer);
begin
  if TXTObject.Name='CHANGEME' then
  begin
     WidthInPix := Xres div 5;
     TXTObject.OnPaint := OnPaintMarker;
     // This line is required:
     PaintObject :=  TXTObject;
  end;
end;

procedure TForm1.OnPaintMarker(Sender : TWPTextObj;
    OutCanvas  : TCanvas;
    XRes, YRes : Integer;
    X, Y, W, H, BASE: Integer );
var o : Integer;
begin
      OutCanvas.Brush.Color := clYellow;
      OutCanvas.RoundRect(x,y,x+w,y+h,XRes div 10, YRes div 10);
      OutCanvas.Pen.Color := clRed;
      o := XRes div 60;
      OutCanvas.Pen.Width := 0;
      OutCanvas.MoveTo(x + o*2, y + h div 2);
      OutCanvas.LineTo(x + w - o*3, y + h div 2);
      OutCanvas.MoveTo(x + w - o * 6, y + h div 2 - o * 3);
      OutCanvas.LineTo(x + w - o*3, y + h div 2);
      OutCanvas.LineTo(x + w - o * 6, y + h div 2 + o * 3);
end;
```

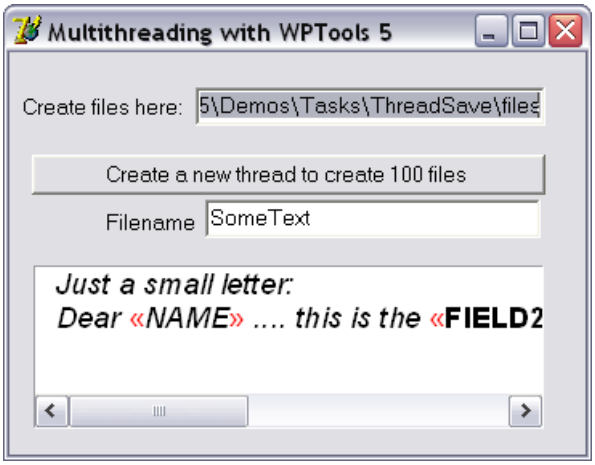
The example above will create the yellow maker:



5.3.27 Working with multiple threads

Unlike many other editor components WPTools 5 can work threadsavely. This makes sense if you use WPTools to create documents. Since WPTools can be used to create and print documents without any window handle beeing required You can use WPTools as a powerful engine to created electronic documents. This includes RTF documents, HTML documents and, with our product wPDF, also PDF documents.

The demo 'ThreadSave' shows how to create a seperate task to do merge text. The merge template is sent from the main thread to the sub thread. Each subthread merges the text 100 times and saves each resulting file as RTF FILE:



The constructor is the most important part of the thread class:

```
constructor TWPToolsThread.Create(const SomeText, DirName, Text: string; Count: Integer);
begin
  inherited Create(false);
  ForceDirectories(DirName);
  RichText := TWPCustomRtfEdit.CreateDynamic;
  {$IFNDEF NOENVIROMENT}
  Enviroment := TWPToolsEnviroment.Create( nil);
  Enviroment.Assign(GlobalWPToolsCustomEnviroment);
  RichText.Memo.RTFData.RTFProps.Enviroment := Enviroment;
  {$ENDIF}
  RichText.OnMailMergeGetText := DoMailMergeGetText;

  RichText.AsString := Text;
  FCount := Count;
  FSomeText := SomeText;
  FDirName := DirName;
end;
```

With this line the editor which is used for the process is created:

```
  RichText := TWPCustomRtfEdit.CreateDynamic;
```

TWPCustomRtfEdit is defined in unit WPCTRMEMO - it is the basic editor class. This class does not contain all features TWPRichTExt has, cannot be attached to a TWPRuler or a TWPToolBar but contains all procedures which are required to create text and tables, insert images and to do mail merge.

The optional code

```
  Enviroment := TWPToolsEnviroment.Create(nil);
  Enviroment.Assign(GlobalWPToolsCustomEnviroment);
  RichText.Memo.RTFData.RTFProps.Enviroment := Enviroment;
```

is only required if you need threadsave printing or if you need to add different object and file format handling classes to the enviroment.

Example C++Builder code to work with dynamic WPTools editor

```
TWPCustomRtfEdit *wp2;
wp2 = new TWPCustomRtfEdit();  // = CreateDynamic
wp2->_MakeDynamic();

wp2->InputString("Hello World\rNext Line",0);

// Insert the text into a differen editor
WPRichText1->SelectionAsString = wp2->AsString;

// If we need to print we need ReformatAll
```

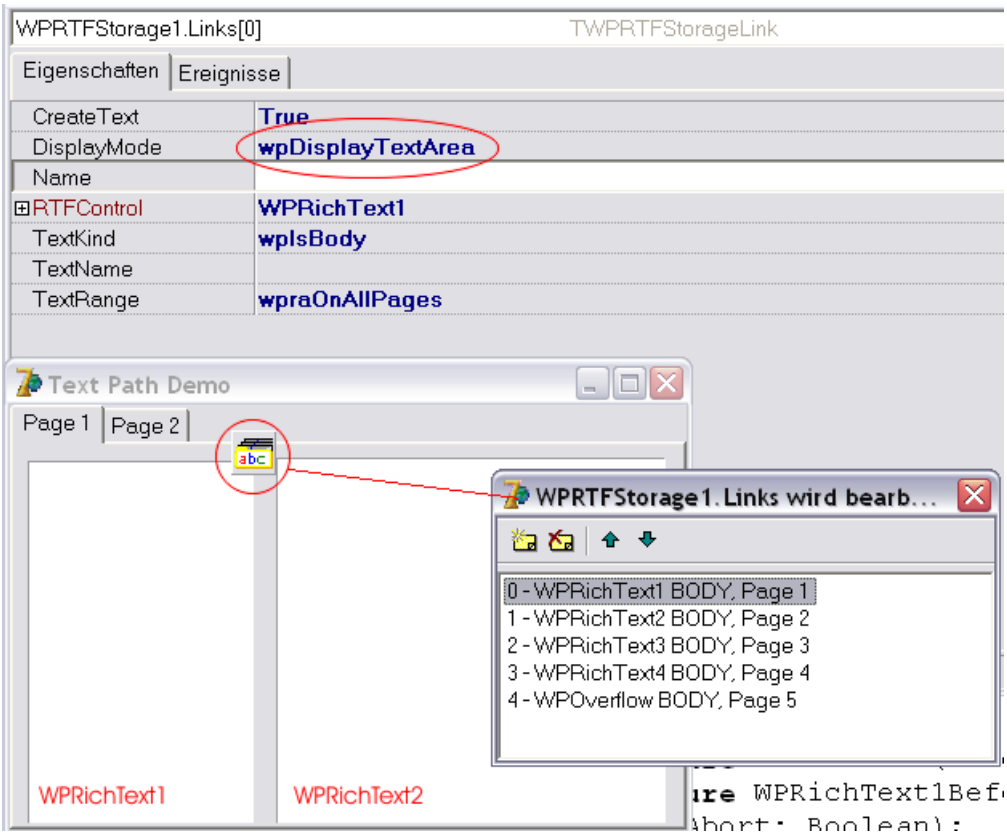
```
wp2->ReformatAll(false,false);
   wp2->PrintPages(1,1);

   // Delete the object
   delete wp2;
```

Note: do not create editor windows which should work interactively using **`CreateDynamic`**!

5.3.28 Create Text Paths

To create text paths simply use the TWPRTFStorage component.

In its property 'Links' you have to create an item for each TWPRichText in the chain.



The following initialization is required:

```
procedure TWPTextPathDemo.FormCreate(Sender: TObject);
begin
 // All path objects need the window handle - otherwise we cannot
 // use our broadcasting system
 WPRichText1.HandleNeeded;
 WPRichText2.HandleNeeded;
 WPRichText3.HandleNeeded;
 WPRichText4.HandleNeeded;

 WPRichText1.InputString('This is a text path. Please create new pages with Ctrl+CR'+#13);
 WPRichText1.CPPosition := MaxInt;
 // Settings for all TWPRichText
 WPRichText1.EditBoxModes := [wpemLimitTextWidth,wpemLimitTextHeight];
 WPRichText1.EditOptions := [wpNoHorzScrolling,wpNoVertScrolling];

 WPRichText2.EditBoxModes := [wpemLimitTextWidth,wpemLimitTextHeight];
 WPRichText2.EditOptions := [wpNoHorzScrolling,wpNoVertScrolling];

 WPRichText3.EditBoxModes := [wpemLimitTextWidth,wpemLimitTextHeight];
 WPRichText3.EditOptions := [wpNoHorzScrolling,wpNoVertScrolling];

 WPRichText4.EditBoxModes := [wpemLimitTextWidth,wpemLimitTextHeight];
 WPRichText4.EditOptions := [wpNoHorzScrolling,wpNoVertScrolling];
end;
```

In the BeforeEditBoxNeedFocus event we use this code:

```
procedure TWPTextPathDemo.WPRichText1BeforeEditBoxNeedFocus(Sender: TObject;
  var Abort: Boolean);
begin
  PageControl1.ActivePageIndex := 0;
end;

procedure TWPTextPathDemo.WPRichText3BeforeEditBoxNeedFocus(Sender: TObject;
  var Abort: Boolean);
begin
  PageControl1.ActivePageIndex := 1;
end;
```

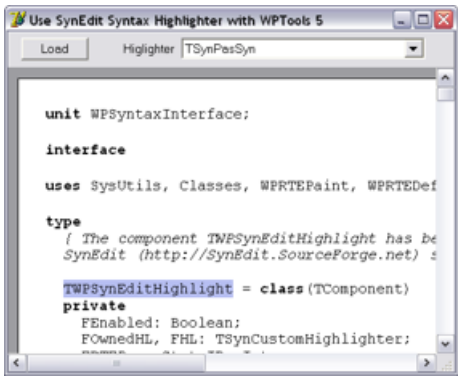
In the OnMouseDown event of the page control we remove the focus from the editors:

```
procedure TWPTextPathDemo.PageControl1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  // The WPRichText must loose the focus - otherwise it is not possible
  // to switch to a different page
  PageControl1.SetFocus;
end;
```

5.3.29 Syntax Highlighting

The component **TWPSynEditHighlight** has been created to use the SynEdit (http://SynEdit.SourceForge.net) syntax highlighters (available for pascal, c++, java, SQL, XML and many many more) with WPTools 5.

Please check out demo SynHighlight.



TWPSynEditHighlight works in an optimized way - it updates the text attributes only when necessary and caches the text attribute ids. The relatively short sourcecode (WPSyntaxInterface) is really worth to be read.

5.3.30 Find Text

To search for text You can use the 'Finder'. The finder is a class which contains several properties to adjust how finding works. It is also able to find using a wildcard, but the found text always has to be in one paragraph.

Please look up the properties of the finder in the online help under 'TWPTextFinder'

The "Finder" demo shows how to create hyperlinks and how to replace colored words. It also includes some demo code to change the attribute of text depending on their current attributes - not using the finder but the 'CurrentCharAttr' interface.

(Note: The display of the hyperlink objects has been enabled in the property FormatOptions)

Convert Hyperlink:

```
with WPRichText1.Finder do
begin
  ToStart;
  EndAtSpace := TRUE;
  while Next('http://*') do
  begin
    SelectText;
    WPRichText1.InputHyperlink(FoundText);
  end;
  EndAtSpace := FALSE;
end;
WPRichText1.HideSelection;
```

Replace red words

```
var Finder: TWPTextFinder;
begin
  Finder := WPRichText1.Finder;
  Finder.Clear;
  Finder.ToStart;
  Finder.CharAttr.SetColor(clRed);
  Finder.EndAtWord := TRUE; // "WholeWord" does not work
  Finder.WildCard := '*';
  while Finder.Next('*') do
  begin
    Finder.FoundText := 'Test';
    Finder.FoundAttr.SetColor(clBlack);
  end;
  Finder.CharAttr.Clear;
  WPRichText1.Refresh;
end;
```

5.3.31 CPChar, CPMoveNext, etc.

WPTools makes it easy for you to loop through all the characters to check for attributes, change attributes or extract or modify text.

Example: Change color of bold text (from Finder demo)

```
  WPRichText1.AttrHelper.Clear;
  WPRichText1.AttrHelper.SetStyles([afsBold]);
  WPRichText1.CPPosition := 0;
  repeat
    if WPRichText1.CurrentCharAttr.Contains(
      WPRichText1.AttrHelper) then
      WPRichText1.CurrentCharAttr.SetColor(clRed);
  until not WPRichText1.CPMoveNext;
  WPRichText1.Refresh;
```

Example: Assign the bold attribute to the selected text

```
var i : Integer;
begin
   i := WPRichText1.SelLength;
   WPRichText1.CPPosition := WPRichText1.SelStart;
   while i>0 do
   begin
     WPRichText1.CPAttr.BeginUpdate;
     WPRichText1.CPAttr.IncludeStyle(afsBold);
     // other changes ...
     WPRichText1.CPAttr.EndUpdate;
     if not WPRichText1.CPMoveNext then break;
     Dec(i);
   end;
end;
```

Instead of this complicated code you can also use
CurrAttr.AddStyle([afsBold])
but the above let you decide for each character which style has to be set.

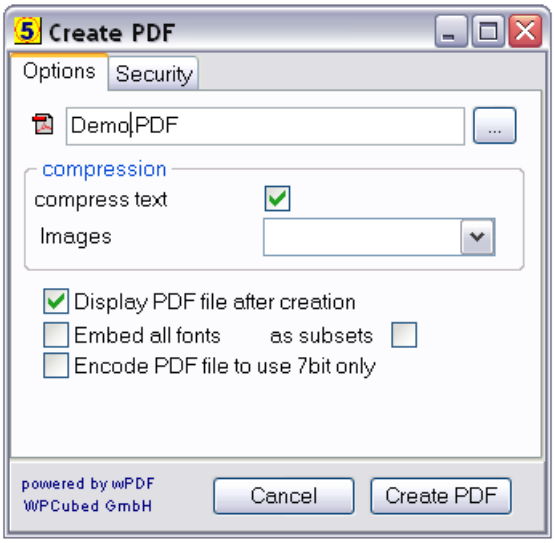
Upgrade notes:

CPChar and CPAttr cannot be used as pointers anymore.

Instead CPChar is a property and CPAttr is an object with properties to manipulate the corresponding TAttr value.

5.3.32 PDF export with wPDF

You can use the provided PDF creation dialog to create a full featured PDF file with our product wPDF.



To display it only these lines are required:

```
uses WPToPDFDlg;

var pdfcreate: TWPCreatePDF;
begin
  pdfcreate := TWPCreatePDF.Create(Self);
  pdfcreate.EditBox := WPRichText1;
  try
    pdfcreate.ShowModal;
  finally
    pdfcreate.Free;
  end;
end;
```

Alternatively this simple code can be used to export PDF files from WPTools 5.

The drawing code used by WPTools 5 will makes sure that the hyperlinks and bookmarks are exported to PDF as they are with the 'old' WPPDFExport component. Please note that the component WPRichText1 is a TWPRichText component created on the form with the property *Visible* set to FALSE. All other properties have not been changed.

```
// uses WPRTEPaint, WPPDFR1, WPPDFR2;

procedure TForm1.ExportToPDF(Sender: TObject);
var WPPDFPrinter1: TWPPDFPrinter;
    i,w,h : Integer;
begin
   WPPDFPrinter1 := TWPPDFPrinter.Create(nil);
   WPPDFPrinter1.FileName :='c:\wptools5demo.pdf';
   WPPDFPrinter1.CompressStreamMethod := wpCompressFastFlate;
   WPPDFPrinter1.AutoLaunch := TRUE;
   WPPDFPrinter1.BeginDoc;
   try
    i := 0;
    while i<WPRichText1.CountPages do
    begin
         w := WPRichText1.Memo.PaintPageWidth[i];
         h := WPRichText1.Memo.PaintPageHeight[i];
         WPPDFPrinter1.StartPage( w, h, Screen.PixelsPerInch, Screen.PixelsPerInch, 0);
       try
         // Use 0 as w and h to let the function calculate the width and height
         WPRichText1.Memo.PaintRTFPage(i,0,0,0,0,WPPDFPrinter1.Canvas, [] );
```

```
finally
        WPPDFPrinter1.EndPage;
      end;
      inc(i);
   end;
  finally
   WPPDFPrinter1.EndDoc;
   WPPDFPrinter1.Free;
  end;
end;
```

To create watermarks simply add additional code which prints on the WPPDFPrinter1.Canvas.

Or you can easily **print 2 pages on the same PDF page**, just make changes in 4 lines:

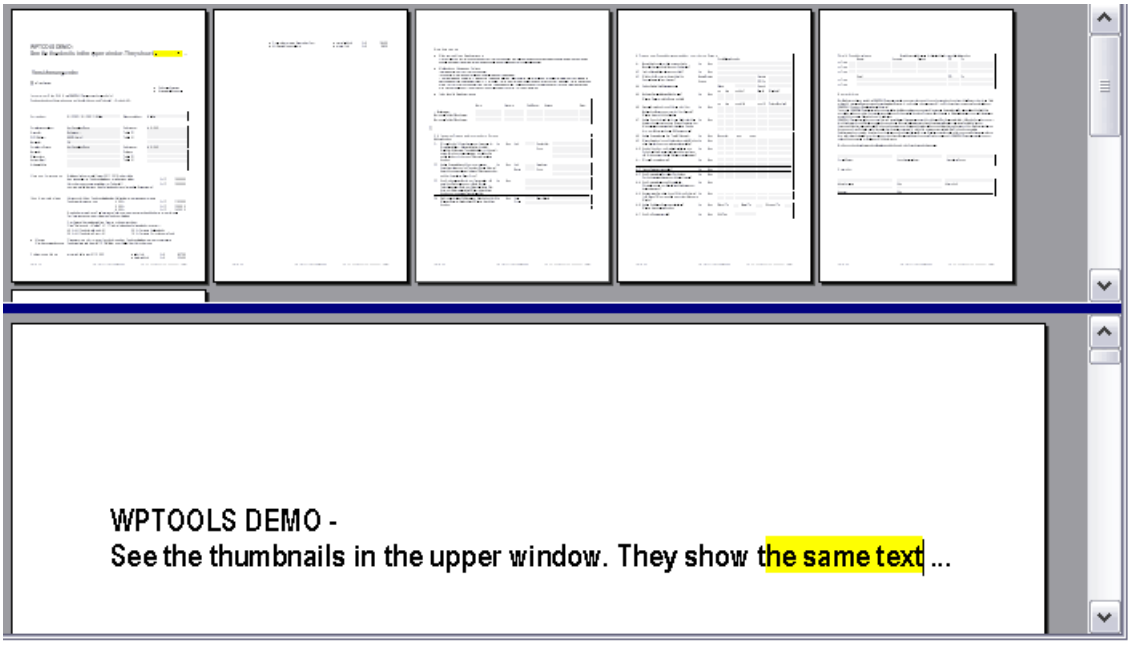
```
var WPPDFPrinter1: TWPPDFPrinter;
    i, w, h : Integer;
begin
   WPPDFPrinter1 := TWPPDFPrinter.Create(nil);
   WPPDFPrinter1.FileName :='c:\wptools5demo.pdf';
   WPPDFPrinter1.CompressStreamMethod := wpCompressFastFlate;
   WPPDFPrinter1.AutoLaunch := TRUE;
   WPPDFPrinter1.BeginDoc;
   try
    i := 0;
    while i<WPRichText1.CountPages do
    begin
         h := WPRichText1.Memo.PaintPageHeight[i];
         w := WPRichText1.Memo.PaintPageWidth[i];
         WPPDFPrinter1.StartPage(w, h div 2,
         Screen.PixelsPerInch, Screen.PixelsPerInch, 0);
       try
         // Use 0 as w and h to let the function calculate the width and height
         WPRichText1.Memo.PaintRTFPage(i,0,0,w div 2,h div 2,WPPDFPrinter1.Canvas, [] );
         WPRichText1.Memo.PaintRTFPage(i+1,w div 2,0,w div 2,h div 2,WPPDFPrinter1.Canvas,
[] );
       finally
         WPPDFPrinter1.EndPage;
       end;
       inc(i,2);
    end;
   finally
    WPPDFPrinter1.EndDoc;
    WPPDFPrinter1.Free;
   end;
end;
```

5.3.33 Multiple Editors for the Same Text

WPTools 5 allows the use of multiple editors which all work with the same text.

This is also known as the split screen technique.

Example:

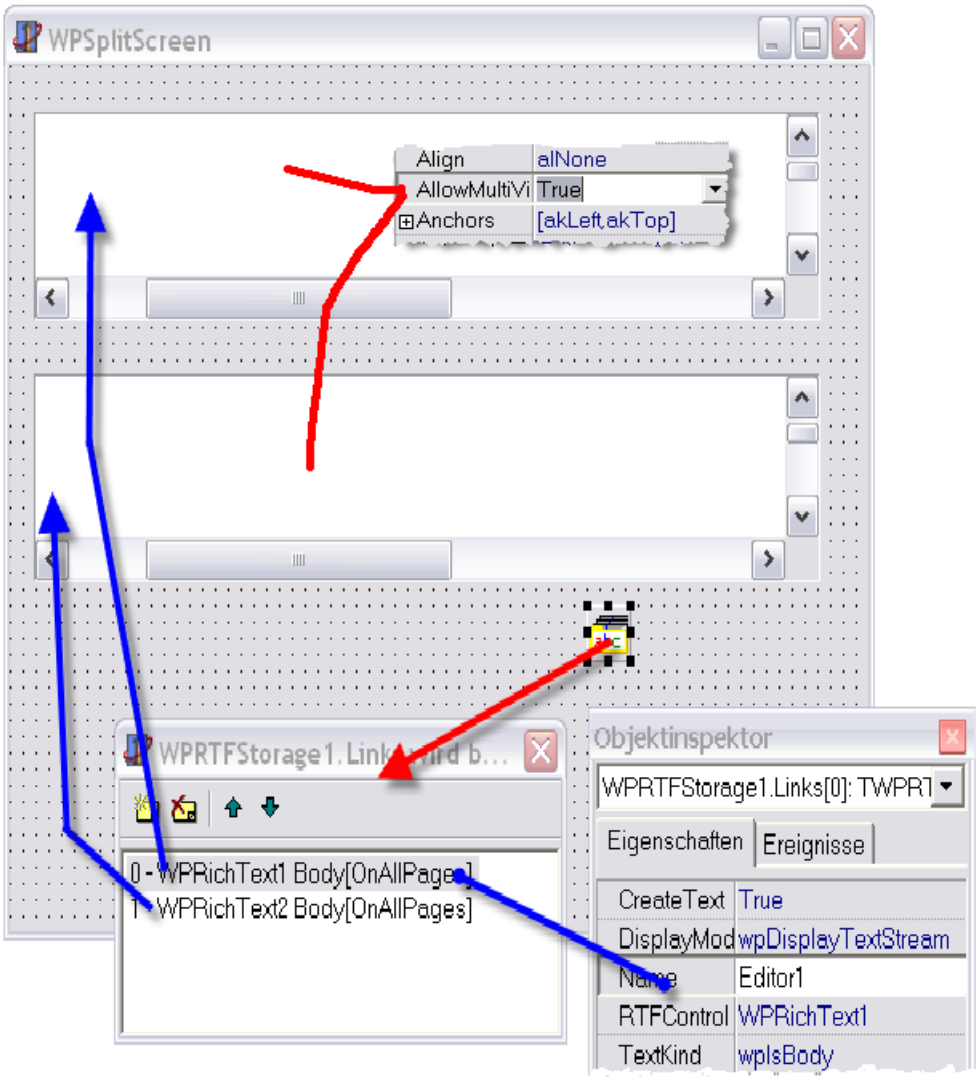


5.3.33.1 Use TWPRTFStorage

The component TWPRTFStorage is used to host the TWPRTFDataCollection for a number of attached editor component.

So if you place several editors on a form you can create Create items in the 'Links' collection and set **wpDisplayTextStream** in property **DisplayMode**.

The property AllowMultiView must be set to TRUE in all editors.



5.3.33.2 Create Multi View in Code

You need an event handler for the event OnInitializeRTFDataObject of the TWPRichText object. This event is executed when the text data is needed the first time. In the event handler you assign a RTFDataCollection which has been created 'outside'. You will need two variables in the TForm class:

```
TForm1 = class(TForm)
...
public
  RTFData: TWPRTFDataCollection;
  RTFDataProps: TWPRTFProps
end;
```

The event handler now initializes the variables and assigns them to any TWPRichText which is using this event handler.

```
procedure TForm1.OnInitRTFData(Sender: TObject;
  var RTFDataObject: TWPRTFDataCollection;
  var RTFPropsObject: TWPRTFProps);
```

```
begin
  if RTFData = nil then
  begin
    RTFData := TWPRTFDataCollection.Create(TWPRTFDataBlock);
    RTFDataProps := TWPRTFProps.Create;
    RTFData.RTFProps := RTFDataProps;
  end;
  RTFDataObject := RTFData;
end;
```

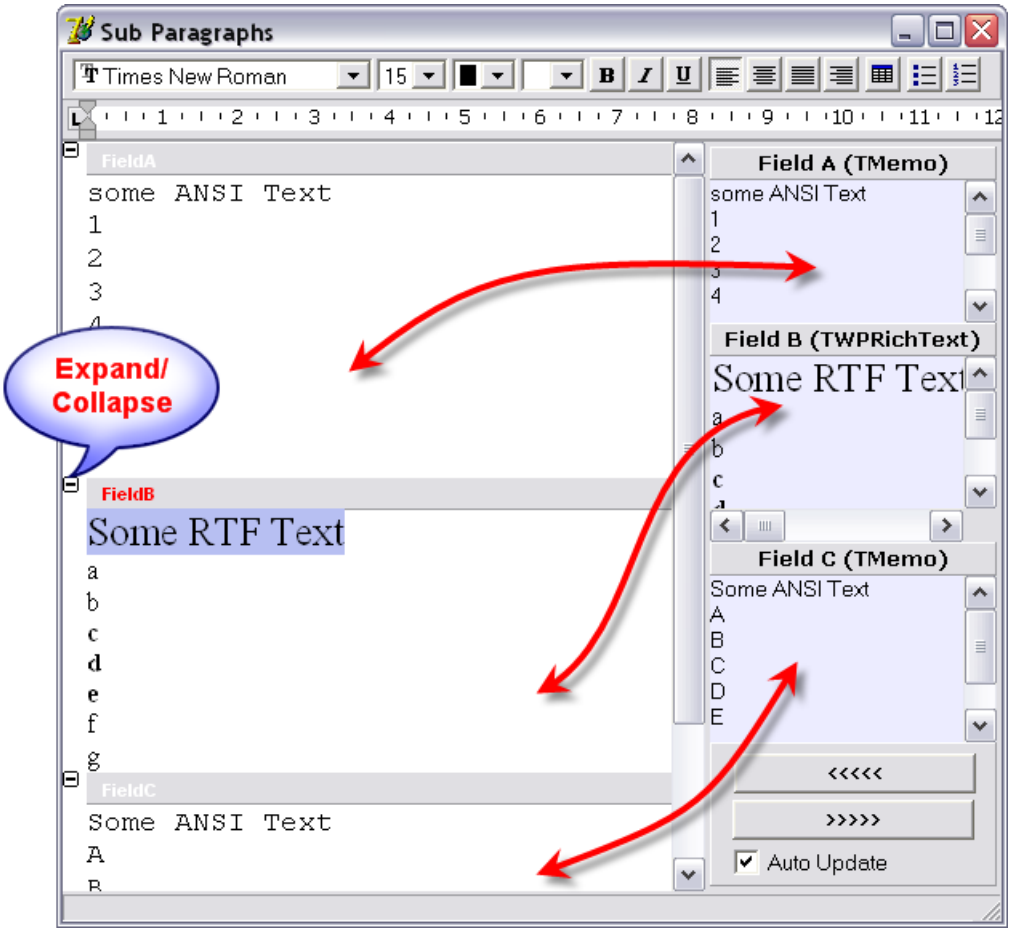
5.3.34 Work with sub paragragraphs

WPTools 5 stores the text in a paragraph tree. The paragraphs are connected using the property 'NextPar'. Some paragraph types require that contents paragraphs are stored on a deeper level, as children paragraphs. For example tables are created that way.

You can use this feature to separate the text into different sections. (Don't mix up with the text sections described here)

These sections can be useful if you need to use one editor to edit or display the text which is stored in different locations, for example different database fields or even database records. It is even thinkabele to use WPTools similar to a database grid!

We created demo which showshow this can be done. You find this demo in the directory SubParagraphs.



The demo shows how to use the paint line event to show the gray header bands. It also contains the code which moves the text from and to the main editor or the fields.

This code is executed when '<<<<' is pressed:

```
procedure TWPSubParDemo.LoadDataClick(Sender: TObject);
var par: TParagraph;
  block: TWPRTFDataBlock;
  mem: TMemoryStream;
  s: string;
  charattr_for_ANSI: Cardinal;
begin
  mem := TMemoryStream.Create;
  AllText.LockScreen;
  try
    AllText.Clear;
   // AFTER the clear we calculate our ANSI character property
    AllText.AttrHelper.Clear;
    AllText.AttrHelper.SetFontName( 'Courier New');
    AllText.AttrHelper.SetFontSize( 1000);
    charattr_for_ANSI := AllText.AttrHelper.CharAttr;
   // and now create the text
    AllText.CheckHasBody;
    block := AllText.ActiveText;
    par := block.FirstPar;
    par.ASet(WPAT_ParProtected, 1);
    par.Name := 'FieldA';
```

```
par.ParagraphType := wpIsXMLTopLevel;
    s := FieldA.Text;
   // SetAllText creates a sub paragraph when the first #13#10 is found!
    if s <> '' then par.SetAllText(#13 + #10 + s, charattr_for_ANSI); // The first ANSI
Text

    par.NextPar := TParagraph.Create(block);
    par := par.NextPar;
    par.ParagraphType := wpIsXMLTopLevel;
    par.ASet(WPAT_ParProtected, 1);
    par.Name := 'FieldB';
    par.ASet(WPAT_CharFontSize, 900);
    FieldB.SaveToStream(mem, 'WPTOOLS');
    mem.Position := 0;
    par.LoadFromStream(mem, 'AUTO', '', []); // The formatted Text

    par.NextPar := TParagraph.Create(block);
    par := par.NextPar;
    par.ParagraphType := wpIsXMLTopLevel;
    par.ASet(WPAT_ParProtected, 1);
    par.Name := 'FieldC';
    s := FieldC.Text;
    if s <> '' then par.SetAllText(#13 + #10 + s, charattr_for_ANSI); // The last ANSI
Text
  finally
    mem.Free;
    AllText.UnLockScreen(true);
  end;
end;
```

This code is executed when '>>>>' is pressed:

```
procedure TWPSubParDemo.PostDataClick(Sender: TObject);
var par, cpar: TParagraph;
  mem: TMemoryStream;
begin
  par := AllText.FirstPar;
  while par <> nil do
  begin
    if par.ParagraphType = wpIsXMLTopLevel then
    begin
      if par.Name = 'FieldA' then
      begin
        FieldA.Text := par.GetAllText(false, false);
      end
      else if par.Name = 'FieldB' then
      begin
        FieldB.LockScreen;
        try
         // This code uses AppendParCopy() to copy the text --------------------
          FieldB.Clear;
          cpar := par.ChildPar;
          while cpar <> nil do
            FieldB.BodyText.AppendParCopy(cpar);
          FieldB.CheckHasBody;

         // The code uses a stream to copy the text ----------------------------
         // this will be useful if you need to save to a blob field!
         {mem := TMemoryStream.Create;
          try
            if par.SaveToStream(mem, true, 'WPTOOLS') then
            begin
              mem.Position := 0;
              FieldB.LoadFromStream(mem, 'WPTOOLS', true);
            end
            else FieldB.Clear;
          finally
            mem.Free;
          end;}
        finally
          FieldB.UnLockScreen(true);
```

```
end;
      end
      else if par.Name = 'FieldC' then
      begin
        FieldC.Text := par.GetAllText(false, false);
      end;
    end;
    par := par.NextPar; // do NOT use "next" here
  end;
end;
```

5.4 BCB Notes

Some notes about the use in C++Builder.

a) Installation

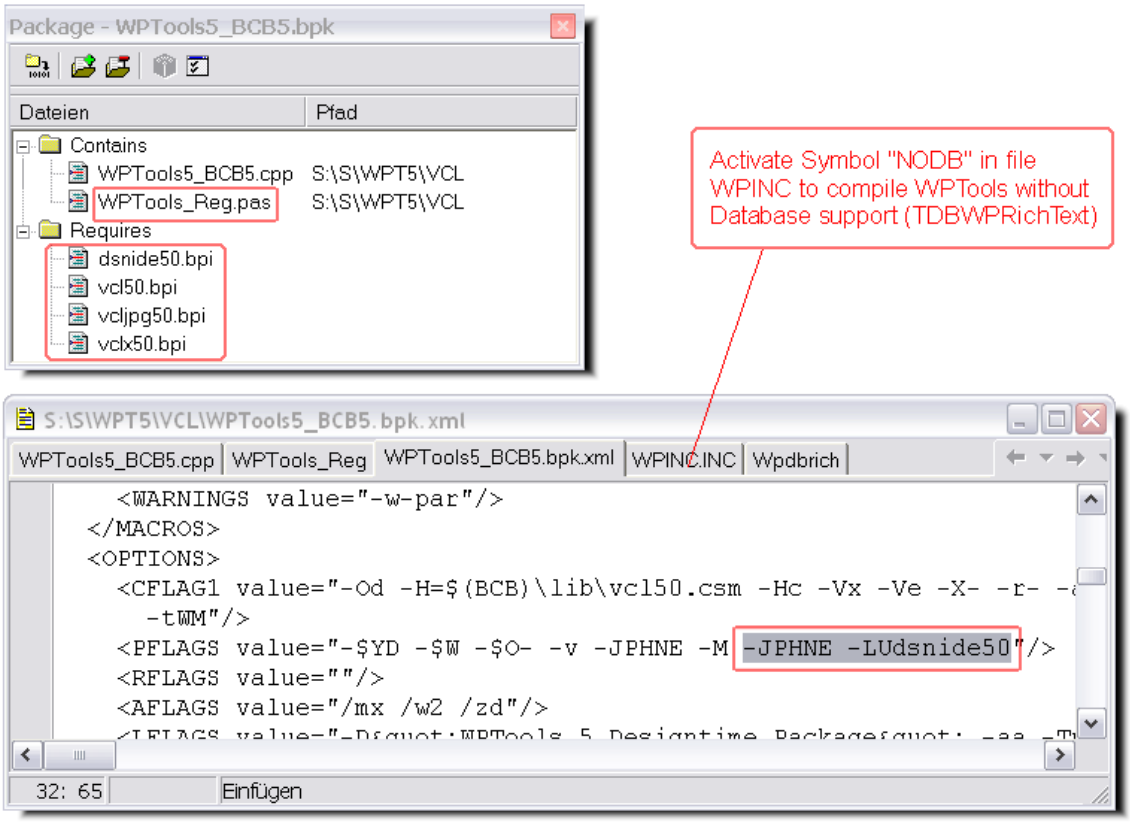
The installation in C++Builder is problematic if you need pre-created package files which cannot match your setup exactly.

So, if there are problems, you can simply install the file WPTools_Reg.PAS as a new component into a new package.

Please make sure the VCLJPG and DSNIDE packages are added. For database support VCLDB is also required. If you have BCB-Standard you can activate the switch NODB in the file WPINC.INC

You will also need to make a change to the 'Option Source', the XML makefile for the package:

When C++Builder compiles the RTFEngine it creates HPP files. In the file WPRTEDefs.HPP it creates 3 lines which are wrong. They start with 'operator' and need to be deleted to make it work.



b) Programming

In general you can use the same techniques in BCB as you use in Delphi

But please note that the dot '.' usually has to be replaced by the arrow '->' to access any objects.

Instead of Txx.Create you usually use ***new***.

Procedures can be called without () in Delphi, for C++ please always add ().

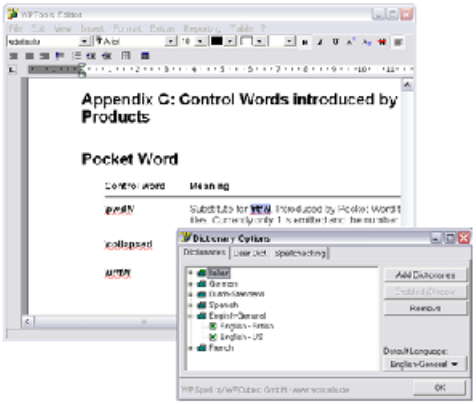
Troubleshooting:

If you get a linker error 'unresolved external' please make sure the WPTools units are found, but only one time (no duplicates), in the **library** and in the **include** path. Please deactivate the use of runtime packages.

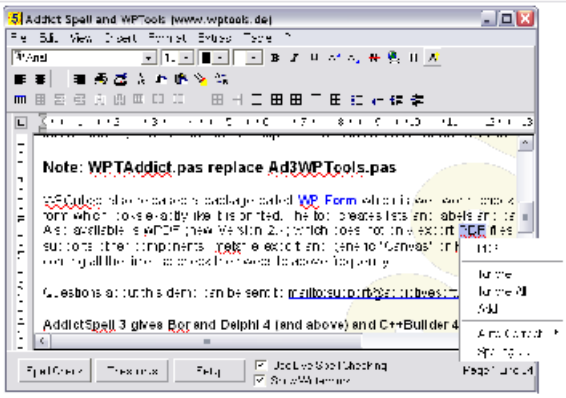
5.5 Adding Spellcheck

WPTools comes with an integrated spell check interface which can be used by the third party products Addict Spell and EDSSpell. (Please see latest partner links)
Our own product WPSpell also uses this procedures and events.

This screen shots show the spell-as-you-go feature with **WPSpell** and **Addict Spell**



WPSpell with Options dialog



Addict Spell and WPTools 5

To activate the spell check interface all you have to do:

- with **Addict Spell**: add unit WPTAddict to the project and use the WPTools spell check actions.
- with **EDSSpell**: add unit eds_wptools to the project
- with **WPSpell**: (registered version. (order link: single, SITE)
    1. Drop the component TWPSpellController  (activate symbol WPSEPLL in file WPINC.INC to compile it into the WPTools package)
    2. Set the property WPSpellController1.Active to TRUE
    3. Add the unit wpspell_link to the uses clause
    4. Use the command WPRichText1.StartSpellCheck() to start/stop spellcheck
- with **WPSpell**: (demo version)
  Just add the unit wpspell_link to the uses clause. You cannot create an instance of the spellcheck controller since it loaded at runtime from the demo DLL.

Advantages of WPSpell

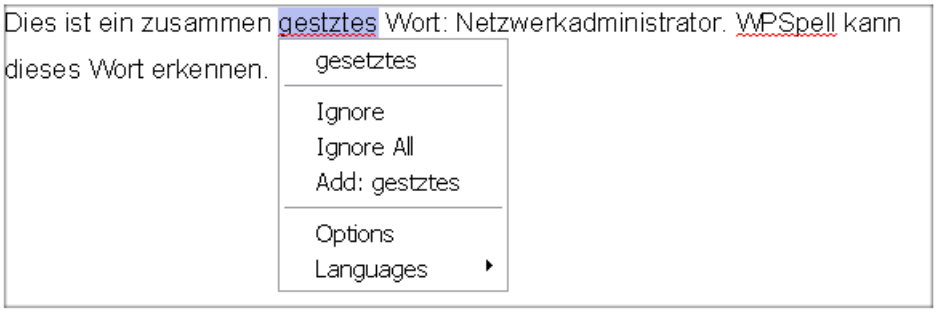
- Spellcheck while typing
- WPSpell has been esspecially tailored to work with WPTools.
- Traditional spellcheck dialog
- Support for spellcheck during input (curly underlines + popup dialog)
- With WPTools 5: instant update of spellcheck markers when switching languages
- Use multiple dictionaries
- Low overhead - dictionaries are not completely loaded into memory (although this option exists, too)
- Setup information automatically stored in INI file or registry
- Dictionary compiler included
- Optional compond word checking with German dictionary
- Very fast dictionary routines

- Complete source code for spellcheck engine included with registered version. The engine has been rewritten to make best use of the latest compiler technology.

In the editor two properties are of interest for spellchecking:

SpellCheckStrategie, possible values are wpspCheckInInit (default), wpspCheckInPaint and wpspCheckInInitAndPaint. We recommend to use wpspCheckInInitAndPaint with WPSpell.

ViewOption flag wpTraditionalMisspellMarkers: If this flag is active instead of the big underlines (~~~) smaller lines are drawn under misspelled words:



5.6 WPReporter Addon

WPReporter is part of WPTools Bundle, WPTools Professional Bundle and WPTools *Premium*.

With WPReporter you get a powerful reporting tool which is tightly integrated into the word processor. The resulting report is a text which can be edited. It creates the reports by merging data into a template with the ability to loop parts of the template (bands).

Also included is a component to add calculation to tables, also for dynamically calculated fields to display subtotals in headers, footers, header-rows and footer-rows.

WPReporter is able to create tables with header and footer rows and sections if you need different page formats in a document.

The following classes are included:

TWPEvalEngine	This class implements the support for formulas.
TDBWPEvalEngine	This class adds database access to TWPEvalEngine
TWPFormulaInterface	This class creates a link between an editor and an TWPEvalEngine to support calculation in text and tables
TWPSuperMerge	This is the main class of WPReporter. It implements the logic to create a new text from data and template texts.

Order link to the upgrade WPTools Standard or WPTools Standard PRO WPTools "Bundle"

5.6.1 Calculation in Text and Tables

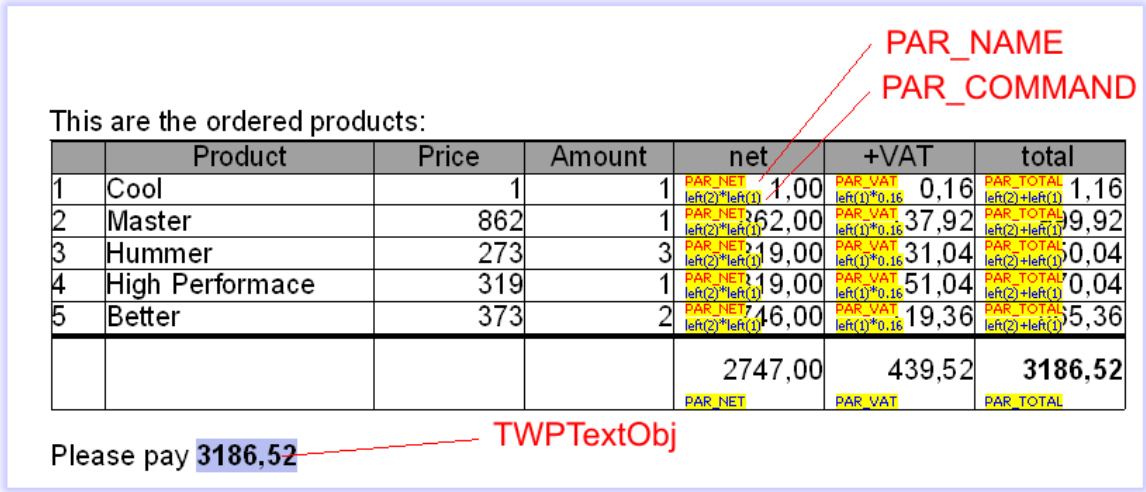
When you have licensed WPReporter you can use powerful calculation commands in WPTools. To use the calculation names and formulas add the component TWPFormulaInterface to the application.

The calculation tool uses names and formulas. Names can be assigned to paragraphs or cells only.
Formulas can be assigned to paragraphs (or cells) and also special TWPTextObj objects. When the text is calculated the paragraph text or the displayed text of the TWPTextObj will show the floating point result of the calculation.

If a formula is just a name (assigned to one or many other paragraphs) the result is the sum of the numbers found in all the paragraph which use this name. This feature makes it easy to sum up values.

Please see the demo TableCalc. This demo includes code to create a simple invoice. It also shows how to activate the optional display of paragraph names and formulas.

Which this option activated you will see an output like this:



You can see that the cells in each column use a different name (highlighted in red). The formula (in blue) uses a relative function left(N) which returns the value of the Nth cell to the left .
The total row uses a formula (in blue) just the name used by the cells which should be summed up.

Please note that this way to calculate is optimized for invoices and similar: The numbers will be always summed as displayed (rounded), not using possible additional decimal values.

The text object which also displays the total is created as simple as:

```
obj := par.AppendNewObject(wpobjTextObject,false,  false);
obj.Name := 'CALC'; // fixed name
obj.Source := 'PAR_TOTAL'; // display the sum of all par with this name
obj.Params := '???'; // initial display text
```

Note: the objects name is 'CALC' which is obligatory.

If you need to create sub totals in header or footer texts You can use a TWPTextObject with the name **PAINT_CALC**.
The 'Source' should be a **+** sign followed by the name of all paragraphs which should be summed

up. Other formulas are not possible since the calculation is not performed by the WPEval unit. This simple calculation is the default action done for the OnTextObjectPaintCalc of the TWPFormulaInterface. Please also see below.

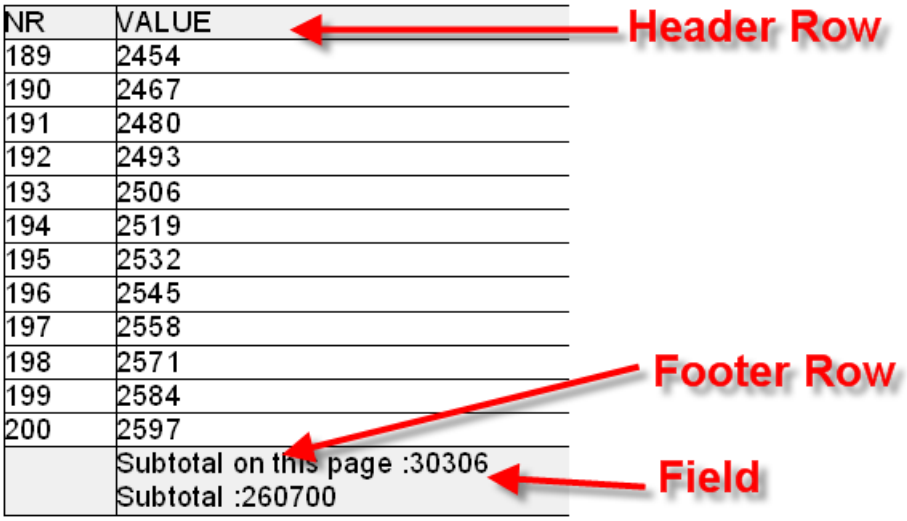
Using HTML syntax such a footer can be created like this:

```
  WPRichText1.HeaderFooter.Get(wpIsFooter,wpraNotOnLastPage).RtfText.AsString  :=
    '<html><div align=right style="border-top-width:0.5pt">Subtotal: <TEXTOBJ
name="PAINT_CALC" source="+PAR_TOTAL">???</TEXTOBJ></div></html>';
```

This functions are created by the unit WPTblCalc and can be used in formulas:

left(N, N2, ...Nn) : sum up the cells to the left
right(N, N2, ...Nn) : sum up the cells to the right
previous(N, N2, ...Nn) : sum up the cells in the same column but previous rows
prior(N, N2, ...Nn) : synonym for previous()
average(name) - calculate the average of all paragraphs with the give name
valcount(name) - count the paragraphs with the give name

The event TWPFormulaInterface. **OnTextObjectPaintCalc** makes it possible to **calculate the contents of fields at paint time**. This is very useful for fields in header or footer texts or repeated table header or footer rows. These fields are not physically duplicated, they are just painted on several pages. So their contents must be calculated at paint time.



The event OnTextObjectPaintCalc provides several parameters which make it possible to evaluate the text on the current page (the page they are painted upon).

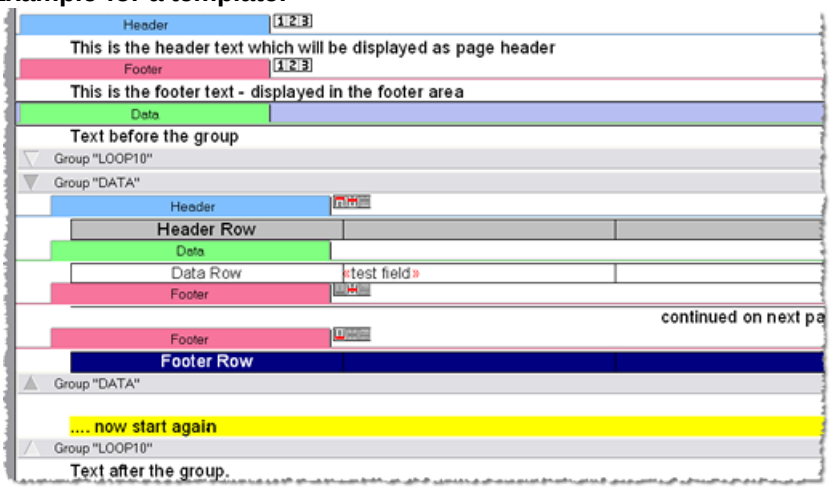
Please see the TableCalc demo. There we sum up all values which are in a certain mail merge field placed on a certain page. A total sum is simply retrieved from a value which is added to each row. We are searching for the last occurance of this value and use it. So if the page break changes and the last row becomes the first row of the next page we use the new last value instead.

5.6.2 Reporting with WPReporter

The reporting creates a new text from a template by mixing in data or calculated text.

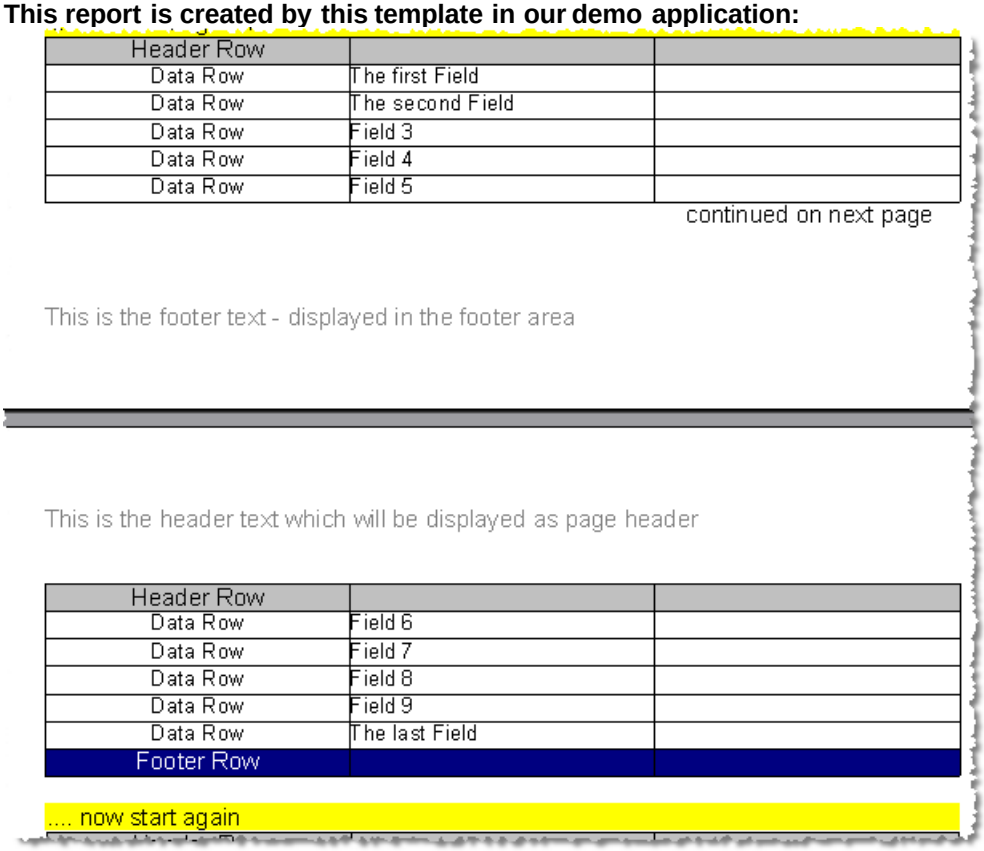
A template consists of text which is sperated into different parts using bands and groups. Unlike groups, bands (data, header or footer) always end with the start of the next band or group. Groups end with the closing of the group - they can also be nested.

Example for a template:



(Hint: You can double-click on the ▽ to collapse any group)

This report is created by this template in our demo application:



Header Row		
Data Row	The first Field	
Data Row	The second Field	
Data Row	Field 3	
Data Row	Field 4	
Data Row	Field 5	

continued on next page

This is the footer text - displayed in the footer area

This is the header text which will be displayed as page header

Header Row		
Data Row	Field 6	
Data Row	Field 7	
Data Row	Field 8	
Data Row	Field 9	
Data Row	The last Field	
Footer Row		

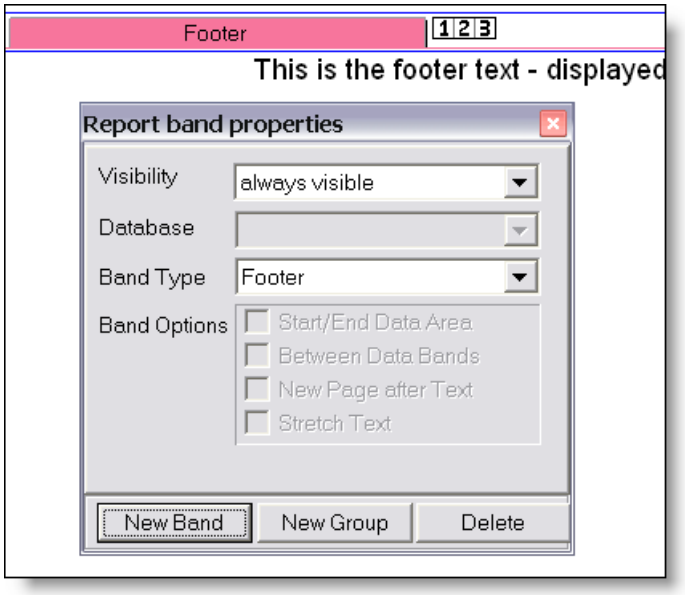
.... now start again

Note that the table uses two footer rows. One is used at the end, the other is used only before a pagebreak. When the created report is saved to RTF MS-Word can use the repeated headers. Unfortunately it does not support repeated footers and hidden rows so both footers will be displayed at the end of the row.

The property SuppressAutomaticHeaderFooter can be used to avoid that header and footers which cannot be properly used with MS Word are created.

Please also see the event **TWPFormulaInterface.OnTextObjectPaintCalc** - it can be used to calculate subtotals for the repeated rows. (See demo TableCalc)

WPReporter includes the band dialog which can be easily used to edit the template:



This dialog is displayed by the **TWPReportBandsDialog** component. Please specify the editor and the *SuperMerge* component. Then use this code to show the dialog:

```
WPReportBandsDialog1.Execute;
```

The buttons "New Band" and "New Group" open popup menus to create new control bands. For groups cou can choose where the group should be created - ie. if the current group should be surrounded by the new one. Please note that while a group or band is selected, pressing **ENTER** creates a new line at the beginning of the contained text. If a group is selected pressing **INSERT** creates a new line AFTER the group. Tip: When text and regular bands (no groups) are selected you can place this text into a new group using "create at the current position".

(Hint: After a band has been added you can use the "Band Type" dialog to change its type. Please also activate one of the check boxes.)

The SuperMerge component (**TWPSuperMerge**) actually does all the work. It combines the text from the source component with the data with the data which was inserted using the OnMailMergeGextText event and sends it to the destination, another memo component of the class TWPRichText.

To attach the two TWPRichText to the TWPSuperMerge please use one line of code to call the procedure **SetSourceDest**. (In WPTools 4 there were 2 properties for this task - they had to be removed due to the different architecture of the new WPReporter)

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   WPSuperMerge1.SetSourceDest(
      SourceText.Memo.RTFData,
      DestText.Memo.RTFData
      );
end;
```

During the creation of the document (inside the destination control) the processing of each band can be switched off and each group can be repeated as often as required.

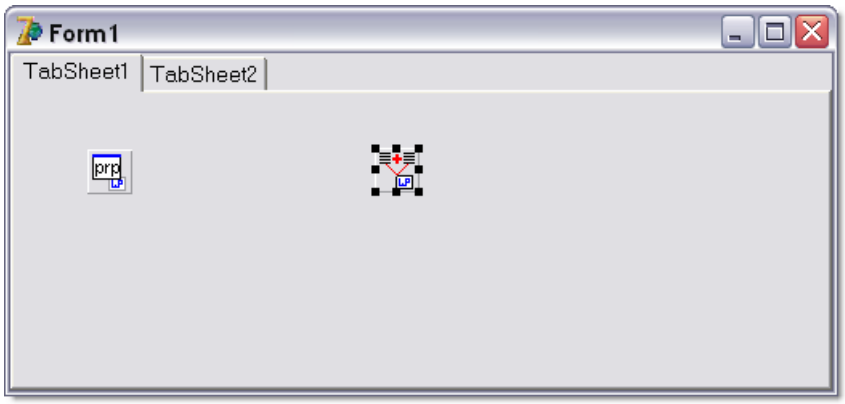
After one paragraph of the template has been copied to the destination control the fields in this paragraphs are replaced by data values. This work is performed by the mail merge function which can be also used without WPReporter. The mailmerge function triggers the OnMaileMergeGetText event for each field which is found in the text. Inside the event the data can be assigned to the object which is used to transfer the data, an image can be inserted or the text format can be changed.

5.6.3 WPReporter - step by step

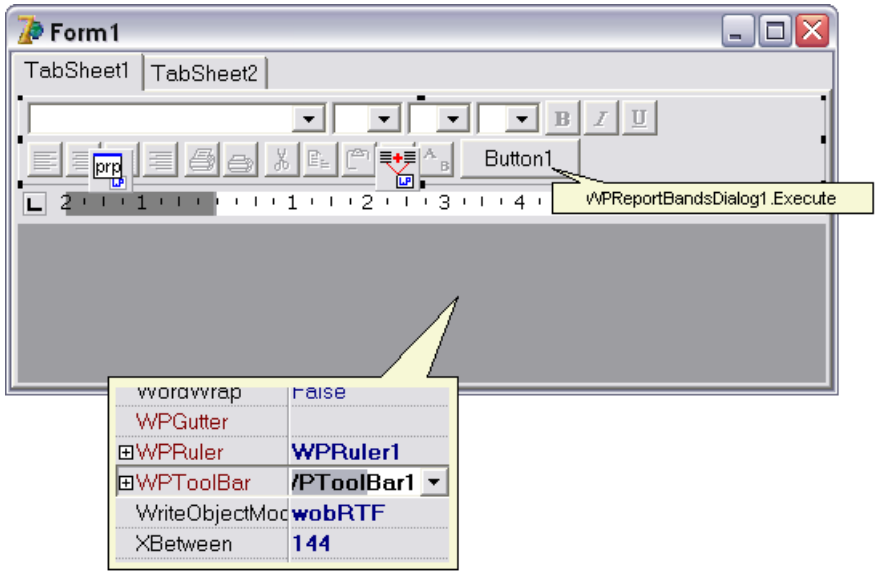
In this chapter we show how to build an application which creates a certain report.

1) Create a form with a page control with 2 pages. The first page will be the template editor, the second page will display the complete report. You can easily hide the first page if you do not want to show your end user the editor.

Also drop a TWPReportBandsDialog and a TWPSuperMerge component.



2) Add a TWPToolBar, TWPRuler, a TWPRichText and also a TButton and connect the controls.



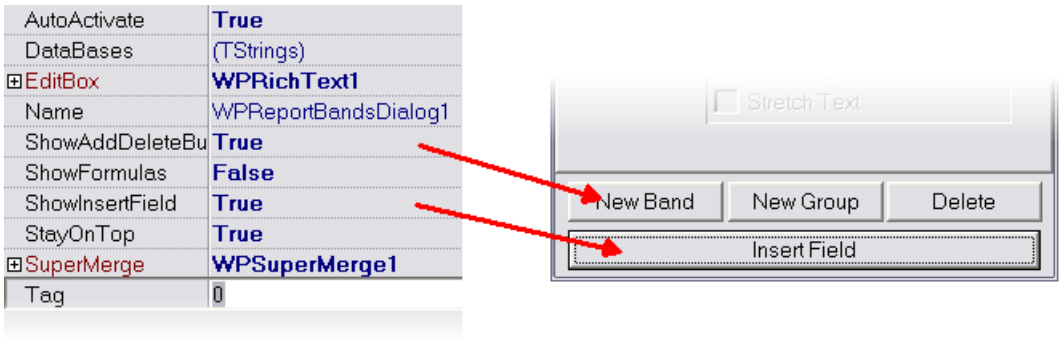
The button is used to show the WPReportBandsDialog1 which is used to add and delete bands.

3) Now please add a second TWPRichText on TabSheet2. We will use this second editor to show the created report. You can also hide the form completely and a TWPPreviewDlg instead. (Property EditBox set to WPRichText2)

4) Now configure the TWPSuperMerge component. This must be done in the OnCreate event of the form.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   WPSuperMerge1.SetSourceDest(
      WPRichText1.Memo.RTFData,
      WPRichText2.Memo.RTFData );
   WPReportBandsDialog1.EditBox := WPRichText1;
end;
```

Also configure the ReportBands dialog:



The property DataBases holds a string list with names which can be selected for the group bands. This names are used select a certain data set or to create and reset a query.

In our example we enter the strings `CUSTOMERS` and `INVOICES`.

5) now its time to create and attach the data bases. To do so we create a separate data module.

(In a real application you would use your existing data module)

For this demo we select the databases CUSTOMERS and ORDERS from the Borland demo database directory and set a master-detail releationship based on the field 'CustNo'.

6) In the event Form.OnShow we fill the field collection of the SuperMerge component. The field list is used to create the drop down menu for the InsertField menu. The collection can be also filled at design time.

```
procedure TForm1.FormShow(Sender: TObject);
var i : Integer;
begin
  WPSuperMerge1.Fields.Clear;
  with WPSuperMerge1.Fields.Add do
  begin
      ParentDatasetName := 'CUSTOMERS';
      ParentDatasetDescription := 'Customer Data';
      RequiredParentGroup := '';
      for i:=0 to DataModule1.CUSTOMER.Fields.Count-1 do
         FieldNames.Add( DataModule1.CUSTOMER.Fields[i].FieldName );
  end;

  with WPSuperMerge1.Fields.Add do
  begin
      ParentDatasetName := 'ORDERS';
      ParentDatasetDescription := 'Orders';
      RequiredParentGroup := 'ORDERS';
      for i:=0 to DataModule1.CUSTOMER.Fields.Count-1 do
         FieldNames.Add( DataModule1.ORDERS.Fields[i].FieldName );
  end;
end;
```

7) Now we want to implement a procedure which creates a generic template which works with our data.

Was add a button: Create Groups and attach this code:

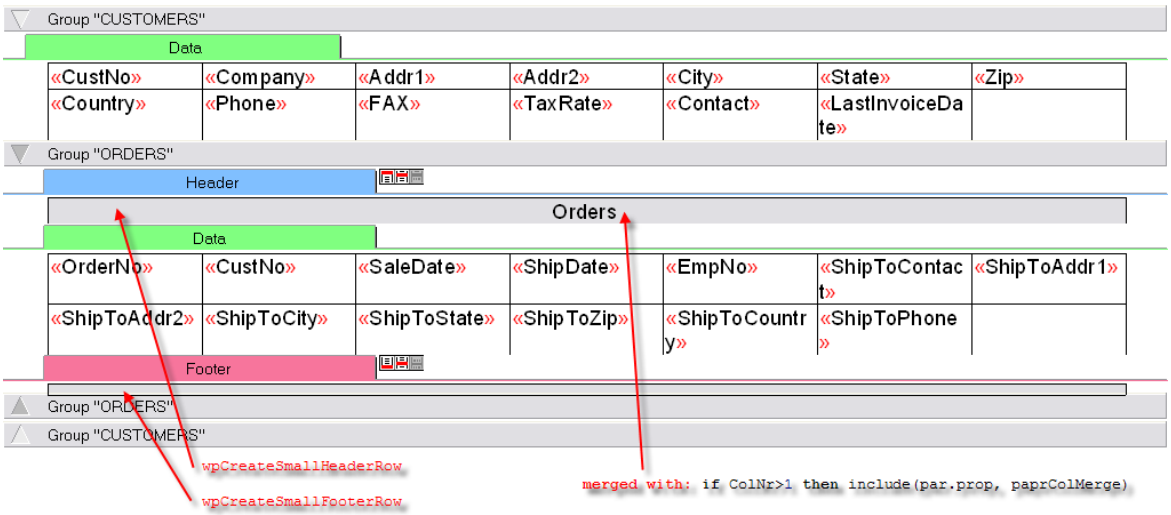
```
procedure TForm1.CreateGroupsClick(Sender: TObject);
begin
  WPSuperMerge1.AddReportGroup('CUSTOMERS',
    [wpCreateBorders,wpCreateDataRow ],
    nil, SetCellStyle, 2);
  WPSuperMerge1.AddReportGroup('ORDERS',
    [wpCreateBorders,wpCreateSmallHeaderRow,wpCreateDataRow,wpCreateSmallFooterRow ],
    nil, SetCellStyle, 2);
end;
```

This code uses the procedure AddReportGroup which creates a group, optional with header and footer rows and with already inserted fields. It accepts a list of fieldnames but can also collect the fields from the field names in the collection Field which we initialized in step (6).

Similar to WTPRichText.TableAdd you can use a callback which will be executed for each created cell. In this callback you can change the text and set properties for the cell. We use this callback.

```
procedure TForm1.SetCellStyle(RowNr, ColNr: Integer; par: TParagraph);
begin
   if (RowNr<0) and ((RowNr and 1)=1) then    // Header Rows
   begin
      par.ASetColor(WPAT_FGColor, clBtnFace);
      if ColNr>1 then include(par.prop, paprColMerge)
      else
      begin
        par.ASet(WPAT_Alignment, Integer(paralCenter));
        par.SetText('Orders');
      end;
   end
   else if (RowNr<0) and ((RowNr and 1)=0) then    // Footer Rows
   begin
      par.ASetColor(WPAT_FGColor, clBtnFace);
      par.ASet(WPAT_SpaceBetween, -100);
      if ColNr>1 then include(par.prop, paprColMerge);
   end;
end;
```

to get this report template as result:



8) Now we can add the logic to actually create the report.

We need an event handler for BeforeProcessGroup. Here we check if the 2 tables are at EOF and reset the client table.

```
procedure TForm1.WPSuperMerge1BeforeProcessGroup(Sender: TWPSuperMerge;
  Band: TWPBand; Count: Integer; var CustomData: TObject; var ProcessGroup,
  IsLastRun: Boolean);
begin
  if Band.Alias='CUSTOMERS' then
  begin
    ProcessGroup := not DataModule1.CUSTOMER.Eof;
    if ProcessGroup then
       DataModule1.ORDERS.First;
  end else
  if Band.Alias='ORDERS' then
  begin
    ProcessGroup := not DataModule1.ORDERS.Eof;
  end;
end;
```

In AfterProcessGroup we move to the next record.

```
procedure TForm1.WPSuperMerge1AfterProcessGroup(Sender: TWPSuperMerge;
```

```
Band: TWPBand; var CustomData: TObject; var Abort: Boolean);
begin
  if Band.Alias='CUSTOMERS' then
  begin
    DataModule1.CUSTOMER.Next;
    Abort := FALSE;
  end else
  if Band.Alias='ORDERS' then
  begin
    DataModule1.ORDERS.Next;
    Abort := FALSE;
  end;
end;
```

Now we need to merge in the data.

We use the event OnMailMergGetText. In this event we locate the field in the sepecified dataset and assign the contents. Of course it would be possible to use calculated fields or variables and constants, too!

```
procedure TForm1.WPSuperMerge1MailMergeGetText(Sender: TObject;
  const inspname: String; Contents: TWPMMInsertTextContents);
var f : TField;
begin
  if Contents.DatasetnamePart='ORDERS' then
       f := DataModule1.ORDERS.FindField(Contents.FieldnamePart)
  else f := DataModule1.CUSTOMER.FindField(Contents.FieldnamePart);
  if f=nil then
       Contents.StringValue := '' // undefined!
  else Contents.StringValue := f.AsString;
end;
```

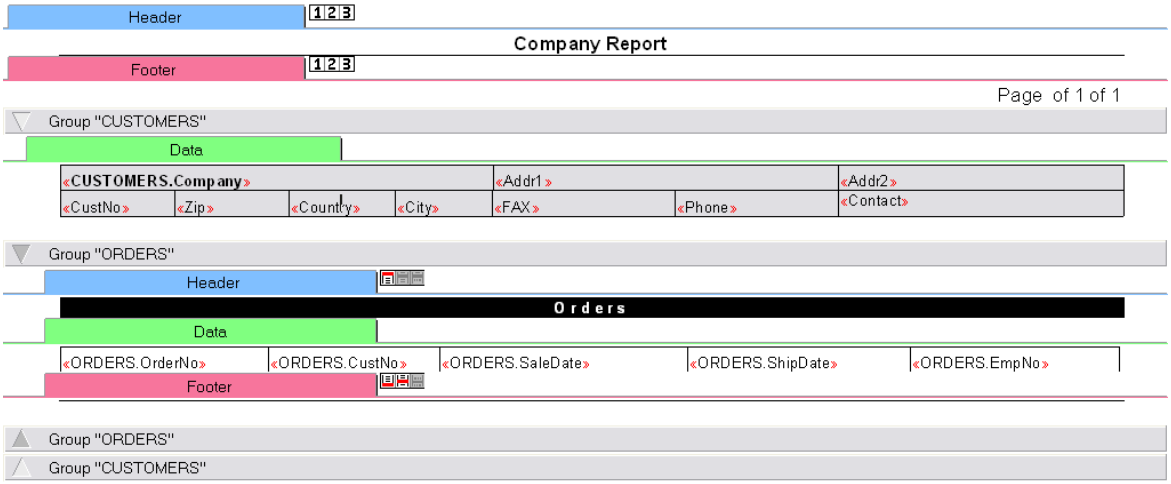
9) Almost complete - we want to show the update report when the active page in the page control is changed

```
procedure TForm1.PageControl1Change(Sender: TObject);
begin
  if PageControl1.ActivePageIndex=1 then // Secod page then
  begin
     DataModule1.CUSTOMER.Open;
     DataModule1.ORDERS.Open;

     DataModule1.CUSTOMER.First;

     WPRichText2.Clear;
     WPRichText2.Header := WPRichText1.Header; // Assign page properties
     WPSuperMerge1.Execute;
     WPRichText2.DelayedReformat;
  end;
end;
```

10) Improve the template

At runtime you can change the reporting template, add fields and also header and footer bands. You can delete rows, move fields, merge cells.

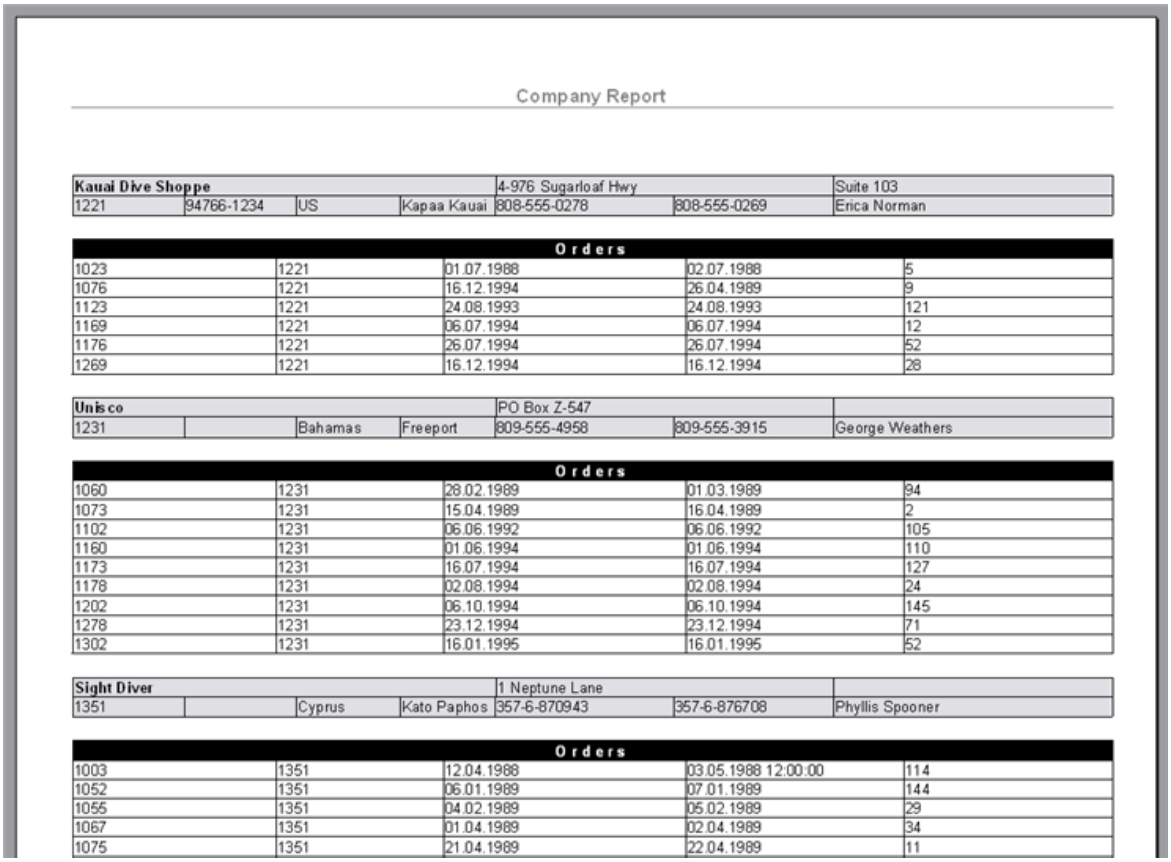
After some changes we got this template - still based on the automatic created template.
Tip: You can save the template as WPT file and load into WPRichText1 when you click on it inside the IDE with right mouse button.



The first, new header and footer bands create page header and footer. The page numbers have been inserted with

```
WPRichText1.InputTextField(wpoPageNumber);
WPRichText1.InoutString(' of ');
WPRichText1.InputTextField(wpoPageCount);
```

This is the created report:



Company Report

Kauai Dive Shoppe				4-976 Sugarloaf Hwy		Suite 103
1221	94766-1234	US	Kapaa Kauai	808-555-0278	808-555-0269	Erica Norman

Orders				
1023	1221	01.07.1988	02.07.1988	5
1076	1221	16.12.1994	26.04.1989	9
1123	1221	24.08.1993	24.08.1993	121
1169	1221	06.07.1994	06.07.1994	12
1176	1221	26.07.1994	26.07.1994	52
1269	1221	16.12.1994	16.12.1994	28

Unisco				PO Box Z-547		
1231		Bahamas	Freeport	809-555-4958	809-555-3915	George Weathers

Orders				
1060	1231	28.02.1989	01.03.1989	94
1073	1231	15.04.1989	16.04.1989	2
1102	1231	06.06.1992	06.06.1992	105
1160	1231	01.06.1994	01.06.1994	110
1173	1231	16.07.1994	16.07.1994	127
1178	1231	02.08.1994	02.08.1994	24
1202	1231	06.10.1994	06.10.1994	145
1278	1231	23.12.1994	23.12.1994	71
1302	1231	16.01.1995	16.01.1995	52

Sight Diver				1 Neptune Lane		
1351		Cyprus	Kato Paphos	357-6-870943	357-6-876708	Phyllis Spooner

Orders				
1003	1351	12.04.1988	03.05.1988 12:00:00	114
1052	1351	06.01.1989	07.01.1989	144
1055	1351	04.02.1989	05.02.1989	29
1067	1351	01.04.1989	02.04.1989	34
1075	1351	21.04.1989	22.04.1989	11

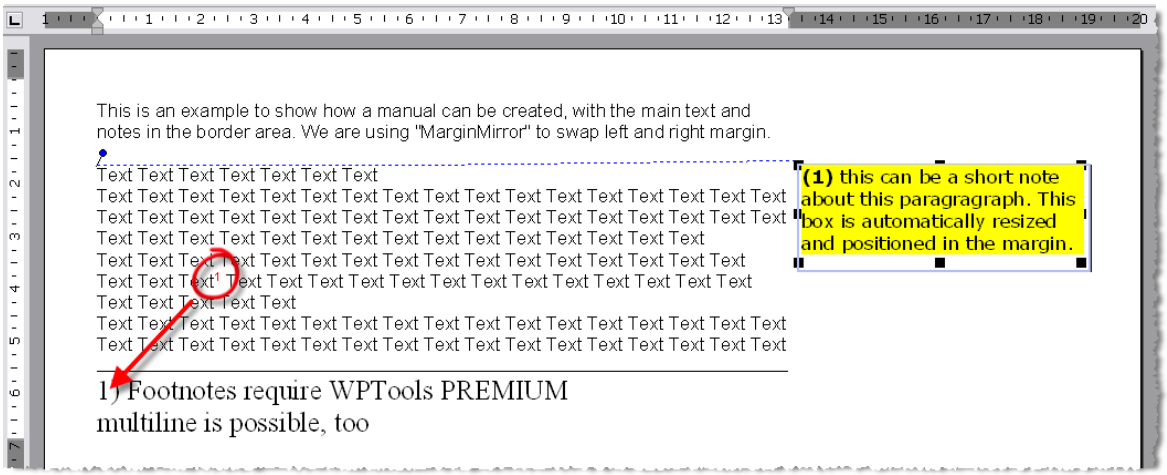
5.7 WPPremium Addon

WPTools Premium adds the support for footnotes, text boxes and (soon) columns to the edition WPTools Professional Bundle. It comes with 100% Source code.

If you have purchased WPTools **Premium** please see the dedicated WPTools Premium manual (PDF) which also includes interesting technical information.

For latest information about WPTools Premium please click here, orderpage.

This text was created by the WPPremium demo. It also uses the "mirror margin" feature, the left and right margin are swapped on each second page.



the footnote in edit mode:

5.7.1 Text boxes

To insert a text box use this code:

```
var obj: TWPORTFTextBox;
  txtobj: TWPTextObj;
begin
  if WPRichText1.CursorOnText.Kind <> wpIsBody then
    ShowMessage('Cannot insert object in Object') else
  begin
    obj := TWPORTFTextBox.Create(WPRichText1);
    obj.WidthTW  := 3000;
    obj.HeightTW := 1000;
    obj.ObjName  := '_AUTO_' + IntToStr(GetTickCount);
    obj.AsString := '<b>Textbox</b> text in HTML, ANSI or RTF format!';
    txtobj := WPRichText1.TextObjects.InsertMovableImage(obj);
    if txtobj <> nil then
    begin
      txtobj.Mode := txtobj.Mode + [ wpobjObjectUnderText, wpobjCreateAutoName ];
      txtobj.RelX := 1440;
      txtobj.RelY := 0;
      // optional
      txtobj.Frame := [wpframe1pt,wpframeShadow];
      WPRichText1.ReformatAll;
      obj.Edit;
      WPRichText1.SetFocus;
    end;
  end;
end;
```

Screenshot of the created box in edit mode (after double click):

Tip: To create a text box which is automatically positioned in the margin of the page use the modes wpobjLockedPos,wpobjPositionInMargin

5.7.2 Footnotes

Please use the method

InputFootnote (PlaceCursor: Boolean; CreateNumber: Boolean = TRUE; InitText: string = #32);

to create a footnote.

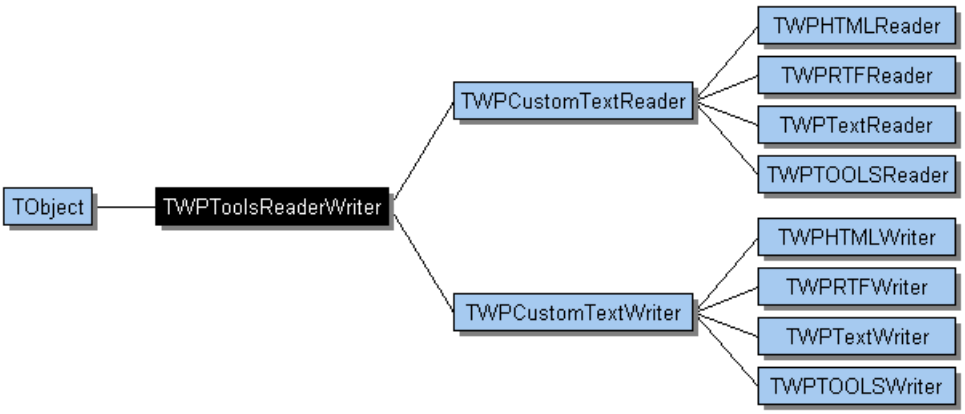
6 Reference

6.1 Reader and Writer

WPTools makes it easy to add support for different file formats.

The import and export is done through a reader and a writer class which must inherit from TWPCustomTextReader and TWPCustomTextWriter.

This basic classes (defined in unit WPRTEDefs) already implement the necessary code for buffered reading and writing.



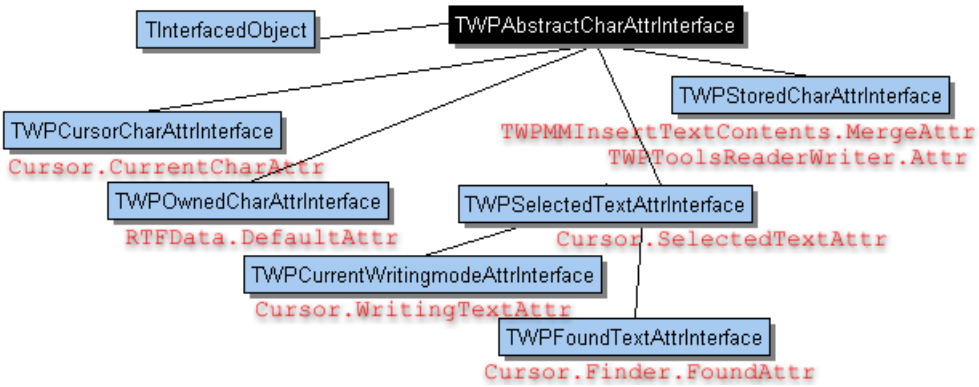
To use a different reader or writer simple use its classname in the property TextLoadFormat or TextSaveFormat or use the alias names, RTF, HTML, ANSI and WPTOOLS.
The reader class also includes a function to check for the format using the first 500 characters of a file, this makes it possible to implement auto detection.

For further reference please see the implementation of the reader and writer classes in the units WPIO*.PAS.

6.2 TWPRTFDataCursor

The cursor class is responsible for cursor movement, text selection and the assignment of properties to the selected or current text.
It has versatile methods to manage markers which are dropped in the text to be collected again later, for example to restore a cursor position or a selection.

The TWPAttrInterface group contains classes to change attributes of different text parts. This chart shows which classes there are and how you can access each of them.



The classes above implement 2 or 3 of the following interfaces (see next chapters).

6.2.1 IWPAttrInterface

The objects which change attributes all implement the interfaces **IWPAttrInterface**, **IWPCharAttrInterface** and, in part, **IWPParAttrInterface**.
This makes it easy to write functions which work with such an interface but do not know if they are changing the selected text or maybe the current text, found text or any other item.

You can, for example, use the following code to change the current paragraph or, if the user has made a selection, the text selected:

```
procedure Form1.SetAllBorders;
  procedure WidthIntDo(const TextInterface : IWPParAttrInterface);
  begin
      TextInterface.ASet(WPAT_BorderFlags,WPBRD_DRAW_All4);
  end;
begin
  if Cursor.IsSelected then
        WidthIntDo( WPRichText1.Memo.Cursor.SelectedTextAttr  )
  else WidthIntDo( WPRichText1.Memo.Cursor.CurrentCharAttr  );
end;
```

To change the current writing mode, please use the interface **WritingTextAttr**. This changes the character mode of the text which is inserted next. If you use it with paragraph attributes it changes the current paragraph.

IWPAttrInterface contains:

```
procedure Clear;
procedure BeginUpdate;
function EndUpdate : Boolean;
```

IWPAttrInterface is implemented by TWPAbstractCharAttrInterface .

6.2.2 IWPCharAttrInterface

The IWPCharAttrInterface contains

```
function GetStyles(var styles: WrtStyle): Boolean;
procedure SetStyles(Value: WrtStyle);
procedure IncludeStyle(Element: TOneWrtStyle);
procedure ExcludeStyle(Element: TOneWrtStyle);
procedure IncludeStyles(Elements: WrtStyle);
procedure ExcludeStyles(Elements: WrtStyle);
function HasStyle(Element: TOneWrtStyle; var Yes: Boolean): Boolean;
procedure SetFont(const FontNr: Integer);
procedure SetFontName(const FontName: string);
function GetFont(var FontNr: Integer): Boolean;
function GetFontName(var FontName: TFontName): Boolean;
function GetFontCharset(var Charset: Integer): Boolean;
procedure SetFontCharSet(const Charset: Integer);
function GetCharSpacing(var DistancePC: Integer): Boolean;
procedure SetCharSpacing(const DistancePC: Integer);
function GetCharLevel(var CharlevelPC: Integer): Boolean;
procedure SetCharLevel(const CharlevelPC: Integer);
function GetCharWidth(var CharwidthPC: Integer): Boolean;
procedure SetCharWidth(const CharwidthPC: Integer);
procedure SetFontSize(Size: Single);
function GetFontSize(var FontSize: Single): Boolean;
procedure SetColorNr(const ColorNr: Integer);
procedure SetColor(const Color: TColor);
procedure SetColorString(const ColorName: string);
function GetColorNr(var ColorNr: Integer): Boolean;
function GetColor(var Color: TColor): Boolean;
procedure SetBGColorNr(const ColorNr: Integer);
procedure SetBGColor(const Color: TColor);
procedure SetBGColorString(const ColorName: string);
function GetBGColorNr(var ColorNr: Integer): Boolean;
function GetBGColor(var Color: TColor): Boolean;
function GetTextLanguage(var Mode: Integer): Boolean;
procedure SetTextLanguage(const mode: Integer);
function GetHighlightMode(var Mode: Integer): Boolean;
procedure SetHighlightMode(const mode: Integer);
function GetTextEffect(var Mode: Integer): Boolean;
procedure SetTextEffect(const mode: Integer);
function GetUnderlineMode(var Mode: Integer): Boolean;
procedure SetUnderlineMode(const mode: Integer);
procedure SetUnderlineColorNr(const ColorNr: Integer);
procedure SetUnderlineColor(const Color: TColor);
function GetUnderlineColorNr(var ColorNr: Integer): Boolean;
function GetUnderlineColor(var Color: TColor): Boolean;
function GetCharStyleSheet(var StyleNr: Integer): Boolean;
procedure SetCharStyleSheet(const StyleNr: Integer);
procedure Clear;
procedure BeginUpdate;
function EndUpdate : Boolean;
```

IWPAttrInterface is implemented by TWPAbstractCharAttrInterface which is the ancestor of all attribute interfaces.

6.2.3 IWPParAttrInterface

The IWPParAttrInterface contains :

```
  function  AGet(WPAT_Code: Byte; var Value: Integer): Boolean;
  procedure ASet(WPAT_Code: Byte; Value: Integer);
  procedure AInc(WPAT_Code: Byte; Offset: Integer; MinValue: Integer = 0);
  procedure ASetColor(WPAT_Code: Byte; Value: TColor);
  procedure ASetAdd(WPAT_Code: Byte; Value: Cardinal);
  procedure ASetDel(WPAT_Code: Byte; Value: Cardinal);
  procedure ASetNeutral(WPAT_Code: Byte; Value: Integer);
  procedure ADel(WPAT_Code: Byte);
  function  TabstopCount: Integer;
  function  TabstopAdd(Value: Integer; Kind: TTabKind;
              Fill: TTabFill; ColorNr: Integer): Boolean;
  function  TabstopDelete(Value: Integer): Boolean;
  procedure TabstopMove(OldValue, NewValue: Integer);
  procedure TabstopClear;
  procedure SetStyle(ParStyleNr: Integer);
  function  GetStyle: Integer;
```

IWPParAttrInterface is implemented by TWPCursorCharAttrInterface and TWPSelectedTextAttrInterface.

Example:
This function is defined in 'CurrAttr' to make it easy to set border attributes.

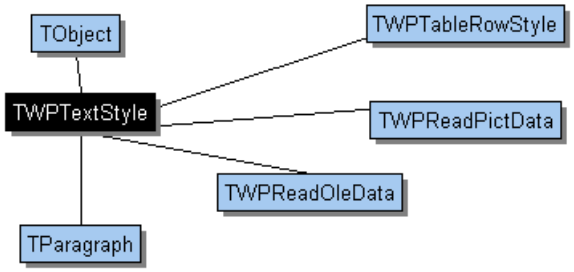
```
procedure TWPSetModeControl.SetBorders(
  LineSelection: TBorderType = [blEnabled, blLeft, blTop, blRight, blBottom];
  WPBRD_mode: Integer = -1;
  ThicknessTW: Integer = -1;
  LeftColor: Integer = -1;
  RightColor: Integer = -1;
  TopColor: Integer = -1;
  BottomColor: Integer = -1;
  AllPadding: Integer = -1;
  DeleteDefaultSettings: Boolean = TRUE);
    // The interface is passed as 'const' - this is very important, otherwise
    // the destructor is called and an AV is thrown
  procedure WidthIntDo(const Int: IWPParAttrInterface);
  var i, brd: Integer;
    procedure SetValue(wpat_code, val: Integer);
    begin
      if val >= 0 then Int.ASet(wpat_code, val)
      else if DeleteDefaultSettings then Int.ADel(wpat_code);
    end;
  begin
    if not (blEnabled in LineSelection) or
      (LineSelection * [blLeft, blTop, blRight, blBottom] = []) then
    begin
      for i := WPAT_BorderTypeL to WPAT_BorderFlags do Int.ADel(i);
    end else
    begin
      brd := 0;
      if blLeft in LineSelection then brd := brd or WPBRD_DRAW_Left;
      if blTop in LineSelection then brd := brd or WPBRD_DRAW_Top;
      if blRight in LineSelection then brd := brd or WPBRD_DRAW_Right;
      if blBottom in LineSelection then brd := brd or WPBRD_DRAW_Bottom;
      Int.ASet(WPAT_BorderFlags, brd);
      SetValue(WPAT_BorderType, WPBRD_mode);
      SetValue(WPAT_BorderWidth, ThicknessTW);
      SetValue(WPAT_BorderColorL, LeftColor);
      SetValue(WPAT_BorderColorT, RightColor);
      SetValue(WPAT_BorderColorB, TopColor);
      SetValue(WPAT_BorderColorR, BottomColor);
      SetValue(WPAT_PaddingLeft, AllPadding);
      SetValue(WPAT_PaddingTop, AllPadding);
      SetValue(WPAT_PaddingRight, AllPadding);
      SetValue(WPAT_PaddingBottom, AllPadding);
    end;
```

```
end;
begin
  if Cursor.IsSelected then
  try
    Cursor.SelectedTextAttr.BeginUpdate;
    WidthIntDo(Cursor.SelectedTextAttr);
  finally
    Cursor.SelectedTextAttr.EndUpdate;
  end
  else
  try
    Cursor.CurrentCharAttr.BeginUpdate;
    WidthIntDo(Cursor.CurrentCharAttr);
  finally
    Cursor.CurrentCharAttr.EndUpdate;
  end;
end;
```

6.3 Manage Style Properties

Most paragraph properties are managed by the functions which are introduced by the TWPTextStyle class.

Please note the inheritance chart of the TParagraph class:



Note: The RTF engine also provides powerful interfaces to manipulate the paragraph and/or character attributes of the current text or the selected text. Both interfaces are part of the **TWPRTFDataCursor** .

6.3.1 List of 'A' methods

The most important 'A' function is:

Method ASet(WPAT_Code: Byte; Value: Integer);

Set the value of a certain property element. Value may be an integer value in the range -8388607 .. +8388607. (24 bits)

Note: If you need to set a color value you can use ASetColor

Reference:

Method **ABorderEqual**

Method **ABorderHash**

Method **AClearCharAttr**

Method **ACopy**

Method **ADel**
This procedure is used to reset a property to the default value.

Method **ADelAllFromTo**
Deletes all properties from/to certain WPAT_values.

Method **ADelAllIn**
Delete all properties which are included in this array.

Method **ADelAttr**

Deletes groups of attributes attributes.

Method **ADeleteDifferentSettings**
This method deletes all properties from this paragraph or style which are not defined in the style which was passed as the parameter. Please also see ADeleteEqualSettings.

Method **ADeleteEqualSettings**
This method deletes all properties from this paragraph or style which are also set in the style which is passed as the parameter. . Please also see ADeleteDifferentSettings.

Method **AGet**
This procedure is used to read most of the paragraph properties. It returns as true if the value was found.

Method **AGetAsINI**

Method **AGetBorder**
Initialize the border record with the setting in this paragraph. When Init=TRUE all values will be initialized.

Method **AGetCharProps**
This procedure fills a TCharAttr record with character properties which are defined here. If Overwrite=FALSE, then it will not overwrite values which are marked as "used" in the mask except for the CharStyle value which is assigned using the 'or' operator.

Method **AGetCharStyle**

Method **AGetColor**

Method **AGetDef**
This procedure is used to read most of the paragraph properties.

Method **AGetDefInherited**
This function reads the attributes defined in this style or paragraph by following the inheritence order. The definition which was made last is used. (See AGetDef)

Method **AGetFontName**
This function retrieves the font name defined for this style or its basestyle (= the inherited WPAT_CharFont property). If no font is defined, the result value is an empty string.

Method **AGetInherited**
This function reads the attributes defined in this style or paragraph by following the inheritence order. The definition which was made last is used. (See AGet)

Method **AGetStringProp**
Reads a property which is a string. Strings are stored as index values into a global string list.

Method **AGetStyleCharAttr**

Method **AGetWPSS**
This function creates a CSS-like style sheet using WPTools special names. Note: All position values are measured in twips. If 'OnlyUsePTag' is set to true, only generic 'P' and 'Tab' tags are created: The string created can then only be used for text which uses the same RTFProps object in the same instance of the application.

Method **AGet_CSS**

Method **AInc**
Increments which increase a property (with offset>0) or decrease a property (with offset<0). In either case the value is checked for the minimum value.

Method **AMerge**
Merges Attributes

Method **ASet**
Set the value of a certain property element. Value may an integer value within the range -8388607 .. +8388607. (24 bits)

Method **ASetAdd**
Executes a bitwise OR operation with the current value and the passed value and assigns the result.

Method **ASetAddCharStyle**
Adds a character style attribute (WPSTY_BOLD...) to the paragraph or style.

Method **ASetAsINI**
Set the propertyies in the WPTools 4 INI format used by the style collection

Method **ASetBaseStyle**
ASetBaseStyle sets the number of the base style for this element. The number is the ID (not the index!) of a style which is stored in the ParStyles property of the TWPRTFProps object.

Method **ASetBorder**
Define the borders using the given border record. Don't forget to set blEnabled in LineType, otherwise all

borders will be disabled. This function overwrites all defined borders in these paragraphs and, if all borders are the same or 0, removes existing definitions.

Method **ASetBorderFlags**

Method **ASetCharStyle**

Method **ASetCharStyle**

Method **ASetColor**
Assigns a color value;to reset to default use clNone.

Method **ASetColorString**
Assigns a color string; to reset to default use an empty string.

Method **ASetDel**
Executes a bitwise AND NOT operation with the current value and the passed value and assigns the result.

Method **ASetDelCharStyle**
Removes a character style attribute (WPSTY_BOLD...) from the paragraph or style. This actually adds a negative style attribute, this means it also deletes an inherited value.

Method **ASetFontName**
This method sets the font used by this style. If an empty string is passed, it deletes the current setting. It cannot be used to set an inherited value which is reported by AGetFontName.

Method **ASetNeutral**
This procedure is used to write most of the paragraph properties. If the value is 0, it will delete an existing entry unless the 0 value is required to override an inherited value. Please note that you may only use 23 bits of Value

Method **ASetStringProp**

Method **ASetWPSS**
This procedure applies the properties defined in the given string. They must have been created using the AGetWPSS function. Please note that white space characters are not expected in the string. If the optional parameter "Merge" is set to TRUE, the name and the base style parameter are not changed (even if they are included in the string) and no properties in the style are initialized before the new properties are applied.

Method **AStringToNumber**
AStringToNumber saves a string to a global list and returns the number to identify the string. This feature is required for style properties which require string variables.

6.3.2 ASet/GetBorder

```
function AGetBorder(var Border: TBorder; IsTableBord, Init, Overwrite: Boolean): Boolean;
```

Initialize the border record with the setting in this paragraph.
When Init=TRUE all values will be initialized.

```
procedure ASetBorder(const Border: TBorder);
```

Define the borders using the given border record. Don't forget to set blEnabled in LineType otherwise all borders will be disabled. This function overwrites all defined borders in these paragraphs and, if all borders are the same or 0, removes existing definitions.

TBorder is defined as:

```
TBorderType = set of (BlLeft, BlTop, BlRight, BLBottom,
    BlInsideV, BlInsideH,   BlDiagLB, BlDiagRB,   BLBar,
    BlFinish, BLBox, BlEnabled );

TWPBrdLine = (brdLeft, brdTop, brdRight, brdBottom,
    brdInsideV, brdInsideH, brdDiagLB, brdDiagRB, brdBar);

TBorder = record
    LineType: TBorderType; // Switch On certain Borders
    BorderType: array[BlLeft..BLBar] of Integer; // Properties of this border
    BorderColor: array[BlLeft..BLBar] of Integer;
    BorderWidth: array[BlLeft..BLBar] of Integer;
    // If This values are <>0 or if "UseAllBorder=true" they
    // are used instead the above arrays
    UseAllBorder: Boolean;
    AllBorderType: Integer;
    AllBorderColor: Integer;
    AllBorderWidth: Integer;
  end;
```

The TBorder record is not saved with the text style. Only the non-default values are stored as regular attributes, using the ASet and AGet procedures using the WPAT_codes.

6.4 WPAT_codes

WPAT_codes are used to set and retrieve paragraph and style attributes. The following groups of codes are defined:

6.4.1 Character Attributes

```
WPAT_CharFont = 1; // the index of the font
  WPAT_CharCharset = 2; // the CharSet of the font
  WPAT_CharFontSize = 3; // FontSize in pt*100
  WPAT_CharWidth = 4; // Character scaling value (display similar to WPAT_CharSpacing)
  WPAT_CharEffect = 5; // Special character effects and character styles
  {*}WPEFF_CUSTOM1 = 1; // wpcustN - 63 Custom tags for whatever fixed styles you want to
develop
  {*}WPEFF_CUSTOMMASK = 63; // The following are bits
  {*}WPEFF_SHADOW = 64; // \shad
  {*}WPEFF_INSET = 128; // \embo
  {*}WPEFF_OUTSET = 256; // \impr
  {*}WPEFF_OUTLINE = 512; // \outl
  {*}WPEFF_ANIMbit1 = 1024; // \animtext1
  {*}WPEFF_ANIMbit2 = 2048; // \animtext2
  {*}WPEFF_ANIMbit3 = 4096; // \animtext4
  {*}WPEFF_ANIMMask = 1024 + 2048 + 4096;

  WPAT_CharStyleMask = 6; // always used with WPAT_CharStyleON to allow the combination of
styles
  WPAT_CharStyleON = 7; // Switch one or multiple of the following Styles on  (WPSTY_BOLD
... )
  {*}WPSTY_BOLD = 1; // Bit 1 bold
  {*}WPSTY_ITALIC = 2; // Bit 2 italic
  {*}WPSTY_UNDERLINE = 4; // Bit 3 underlined (solod)
  {*}WPSTY_STRIKEOUT = 8; // Bit 4 strikeout
  {*}WPSTY_SUPERSCRIPT = 16; // Bit 5 superscript
  {*}WPSTY_SUBSCRIPT = 32; // Bit 6 subscript
  {*}WPSTY_HIDDEN = 64; // Bit 7 hidden text
  {*}WPSTY_UPPERCASE = 128; // Bit 8 all uppercase
  {*}WPSTY_SMALLCAPS = 256; // Bit 9 all uppercase but non-captitals are 20% larger
  {*}WPSTY_LOWERCASE = 512; // Bit 10 all lowercase
  {*}WPSTY_NOPROOF = 1024; // Bit 11 \noproof - disable spellcheck for this
  {*}WPSTY_DBLSTRIKEOUT = 2048; // Bit 12 strikeout - double solid line
  {*}WPSTY_BUTTON = 4096; // button (use background color) - with frame
  {*}WPSTY_PROTECTED = 8192; // protected text - can be optionally handled as shaded text
  {*}WPSTY_USERDEFINED = 16384; // Bit 15 user defined flag
  {* bit 16 must be unused }
  // The following 'styles' are stored as bits in the highest byte of the CharAttr[x]
integer.
  // They are usually applied and removed dynamically and so do not fit  the
TWPCharAttrCache technology
  // They can be added to text using par. AddFlagAttr
  {*}cafsHyphen = $01000000; // assigned by reader or Ctrl+minus, break here
  {*}cafsWasChecked = $02000000; // used by spellcheck routine - don't check again
  {*}cafsMisSpelled = $04000000; // used by spellcheck routine - red line
  {*}cafsMisSpelled2 = $08000000; // used by spellcheck routine - green line
  {*}cafsInsertedText = $10000000; // WPTOOLS PREMIUM: Revision Marks
  {*}cafsDeletedText = $20000000; // WPTOOLS PREMIUM: Revision Marks
  {*}cafsWordHighlight = $40000000; // highlighted by Find Routine
  {*}cafsDelete = $80000000; // Text is marked for deletion  \deleted
  {*}cafsALL = $FF000000;
  {*}cafsNONE = $00FFFFFF;
  WPAT_CharColor = 8; // The text color (as index in palette)
  WPAT_CharBGColor = 9; // The text background color (as index in palette)
  WPAT_CharSpacing = 10; // "Letter-Spacing" in twips, 0..$8000 = EXPAND, $8001- $FFFF =
COMPRESS
  WPAT_CharLevel = 11; // Move Character up or down - in half points (RTF: \up \dn }
  // 0..$8000 = UP, $8001- $FFFF = down
  WPAT_CharHighlight = 12; // {reserved} Highlight mode (different styles and colors)
  WPAT_UnderlineMode = 13; // {reserved} Underlining mode, 0=off, 1=solid, 2=double, 3=
dotted ...
  {*}WPUND_Standard = 1; // Underline Style 1
  {*}WPUND_Dotted = 2;
  {*}WPUND_Dashed = 3;
  {*}WPUND_Dashdotted = 4;
  {*}WPUND_Dashdotdotted = 5;
  {*}WPUND_Double = 6;
  {*}WPUND_Heavywave = 7;
```

```
{*}WPUND_Longdashed = 8;
  {*}WPUND_Thick = 9;
  {*}WPUND_Thickdotted = 10;
  {*}WPUND_Thickdashed = 11;
  {*}WPUND_Thickdashdotted = 12;
  {*}WPUND_Thickdashdotdotted = 13;
  {*}WPUND_Thicklongdashed = 14;
  {*}WPUND_Doublewave = 15;
  {*}WPUND_WordUnderline = 16;
  {*}WPUND_wave = 17;
  {*}WPUND_curlyunderline = 18; // only used for spellcheck
  WPAT_UnderlineColor = 14; // {reserved} Underlining color, 0=text color, otherwise
colorindex +1
  WPAT_TextLanguage = 15; // {reserved} Language of the text
  WPAT_CharStyleSheet = 16; // CharacterStyle
```

6.4.2 Paragraph Attributes

```
/ Margin Group ------------------------------------------------------------
  WPAT_IndentLeft = 17; // Indent Left  (CSS = margin)
  WPAT_IndentRight = 18; // Indent Right (CSS = margin)
  WPAT_IndentFirst = 19; // Indent First (CSS = text-indent)
  WPAT_SpaceBefore = 20; // Space Before (CSS = margin)
  WPAT_SpaceAfter = 21; // Space After  (CSS = margin)
  WPAT_LineHeight = 22; // LineHeight in in % ( Has priority over WPAT_SpaceBetween)
  WPAT_SpaceBetween = 23; // Space Between  (CSS = margin)  - negative : Absolute,
Positive minimum
  WPAT_PaddingLeft = 24; // Distance from Border to Text (CSS = padding) tscellpaddt /
trpaddl
  WPAT_PaddingRight = 25; // Distance from Border to Text (CSS = padding)
  WPAT_PaddingTop = 26; // Distance from Border to Text (CSS = padding)
  WPAT_PaddingBottom = 27; // Distance from Border to Text (CSS = padding)

  // Alignment Group -----------------------------------------------------------
  WPAT_WordSpacing = 28; // Value = 0 for default or % of EM (ignored for justifed text)
  WPAT_Alignment = 29; // paralLeft, ...
  WPAT_VertAlignment = 30; // Cells: paralVertTop, paralVertCenter, paralVertBottom,
paprVertJustified
  // Inline Images ??? : baseline, sub, super, top, text-top,middle,bottom,text-bottom

// ---------------------------------------------------------------------------
  // Numbering - either reference to a different style or defined here
  // For 'old' PN numbering the numbering attributes are found in the paragraphs!
  // ---------------------------------------------------------------------------

  WPAT_NumberSTYLE = 31; // Reference to a different Style. It can be a single style or
  // a style which is part of an outline style sheet. In the letter case the correct
  // style will be be located inside this group using the WPAT_NumberLEVEL parameter.
  WPAT_NumberLEVEL = 32; // Numbering Level
  // The following styles are used for simple paragraph numbering and also inside
  // numbering styles. Numbering styles can also use all other paragraph attributes!
  // Please also note that 'WPAT_NumberLEVEL' on its own does NOT activate numbering.
  // WPAT_NumberSTYLE or WPAT_NumberMODE must be also specified.
  // If only WPAT_NumberLEVEL is used this just specifies the current outline level.
  WPAT_OUTLINELEVEL = 32; // synonym for WPAT_NumberLEVEL
  WPAT_NumberMODE = 33; // Supported elements are: (\levelnfcN)
  {*}WPNUM_ARABIC = 1; // Arabic (1, 2, 3)
  {*}WPNUM_UP_ROMAN = 2; // Uppercase Roman numeral (I, II, III)
  {*}WPNUM_LO_ROMAN = 3; // Lowercase Roman numeral (i, ii, iii)
  {*}WPNUM_UP_LETTER = 4; // Uppercase letter (A, B, C)
  {*}WPNUM_LO_LETTER = 5; // Lowercase letter (a, b, c)
  {*}WPNUM_LO_ORDINAL = 6; // Lowercase Ordinal number (1st, 2nd, 3rd)
  {*}WPNUM_Text = 7; // Cardinal text number (One, Two Three)
  {*}WPNUM_ORDINAL_TEXT = 8; // Ordinal text number (First, Second, Third)
  {*}WPNUM_WIDECHAR = 15; // Double-byte character
  {*}WPNUM_CHAR = 16; // Single-byte character
  {*}WPNUM_CIRCLE = 19; // Circle numbering (*circlenum)
  {*}WPNUM_ARABIC0 = 23; // Arabic with leading zero (01, 02, 03, ..., 10, 11)
  {*}WPNUM_BULLET = 24; // Bullet (no number at all)
  WPAT_NumberTEXTB = 34; // Text Before (index in stringlist of RTFFProps) (obsolete)
  WPAT_NumberTEXTA = 35; // Text After (index in stringlist of RTFFProps)  (obsolete)
  WPAT_NumberTEXT = 36; // Char #1..#10 are the level placeholders, the rest is the
surrounding text
  WPAT_Number_STARTAT = 37; // Start Numbering if this is first in level
  WPAT_Number_ALIGN = 38; // 0=Left, 1=Center, 2=Right
  WPAT_Number_SPACE = 39; // Minimum distance from the right edge of the number to the
start of the paragraph text
  WPAT_NumberFONT = 40; // Optional: Font for the text
  WPAT_NumberFONTSIZE = 41; // Optional: FontSize  (pnfs) in pt * 100
  WPAT_NumberFONTCOLOR = 42; // Optional: FontColor (pncf)
  WPAT_NumberFONTSTYLES = 43; // Optional: CharStyles bitfield ( pnb,  pni, pnul etc ...)
  WPAT_NumberINDENT = 44; // Old Style Indent - NumberStyles may also use the
INDENTFIRST/INDENTLEFT props!
  WPAT_NumberFLAGS = 45; //
  {*}WPNUM_FLAGS_COMPAT = 1; // Compatibility to old RTF
  {*}WPNUM_FLAGS_USEPREV = 2; // Use text from previous level ( pnprev)
```

```
{*}WPNUM_FLAGS_USEINDENT = 4; // Use Indent from previous level
  {*}WPNUM_FLAGS_FOLLOW_SPACE = 8; // A space follows the number, Default = TAB!
  {*}WPNUM_FLAGS_FOLLOW_NOTHING = 16; // nothing follows the number
  {*}WPNUM_FLAGS_LEGAL = 32; // convert previous levels to arabic  (\levellegal=)
  {*}WPNUM_FLAGS_NORESTART = 64; // if this level does not restart its count each time a
number of a higher level is reached
  {*}WPNUM_FLAGS_ONCE = 128; // Number each cell only once in a table
  {*}WPNUM_FLAGS_ACROSS = 256; // Number across rows (the default is to number down
columns)
  {*}WPNUM_FLAGS_HANG = 512; // Paragraph uses a hanging indent
  {*}WPNUM_NONumberING = 1024; // DO NOT NumberATE!
  {*}WPNUM_NumberSKIP = 2048; // Increase number but do not display
  WPAT_NumberPICTURE = 46; // Reserved for number pictures
  WPAT_Number_RES1 = 47;
  WPAT_Number_RES2 = 48;
  WPAT_Number_RES3 = 49;

  // Used for Colors and background  ----------------------------------------
  WPAT_BGColor = 50; // Background Color
  WPAT_FGColor = 51; // Foreground Shading Color
  WPAT_ShadingValue = 52; // Background Shading Percentage in % \tscellpct
  WPAT_ShadingType = 53; // Background Shading Type ( \tsbgbdiag ... )
  {*}WPSHAD_solidbg = 0; // Solid Background - use WPAT_BGColor  and WPAT_ShadingValue
  {*}WPSHAD_solidfg = 1; // Solid Background - use WPAT_FGColor  and WPAT_ShadingValue
  {*}WPSHAD_clear = 2; // Clear Background
  {*}WPSHAD_bdiag = 3; // Backward diagonal pattern.
  {*}WPSHAD_cross = 4; // Cross pattern.
  {*}WPSHAD_dcross = 5; // Diagonal cross pattern.
  {*}WPSHAD_dkbdiag = 6; // Dark backward diagonal pattern.
  {*}WPSHAD_dkcross = 7; // Dark cross pattern.
  {*}WPSHAD_dkdcross = 8; // Dark diagonal cross pattern.
  {*}WPSHAD_dkfdiag = 9; // Dark forward diagonal pattern.
  {*}WPSHAD_dkhor = 10; // Dark horizontal pattern.
  {*}WPSHAD_dkvert = 11; // Dark vertical pattern.
  {*}WPSHAD_fdiag = 12; // Forward diagonal pattern.
  {*}WPSHAD_horiz = 13; // Horizontal pattern.
  {*}WPSHAD_vert = 14; // Vertical pattern.
  WPAT_BGBitMap = 54; // Background Image.
  WPAT_BGBitMapMode = 55; // Bitfield for scroll, center, center, repeat
```

6.4.3 Border Attributes

The TBorder record is not saved with the text style, only the non-default values are stored as regular attributes, using the ASet and AGet procedures. The WPAT_ codes used for border parameters are:

```
  WPAT_BorderTypeL = 60; // Border Mode Left(no, single, double etc)
  WPAT_BorderTypeT = 61; // Border Mode Top (no, single, double etc)
  WPAT_BorderTypeR = 62; // Border Mode Right(no, single, double etc)
  WPAT_BorderTypeB = 63; // Border Mode Bottom (no, single, double etc)
  WPAT_BorderTypeDiaTLBR = 64; // Diagonal Line - TopLeft/BottomRight ( \cldglu )
  WPAT_BorderTypeDiaTRBL = 65; // Diagonal Line - TopRight/BottomLeft
```

The following constants are used as values for the 'Type' attribute

```
  {*}WPBRD_SINGLE = 0; // \brdrs      Single-thickness border. (=Default)
  {*}WPBRD_NONE = 1; // \brdrnil \brdrtbl  No Border
  {*}WPBRD_DOUBLEW = 2; // \brdrth    Double-thickness border.
  {*}WPBRD_SHADOW = 3; // \brdrsh     Shadowed border.
  {*}WPBRD_DOUBLE = 4; // \brdrdb     Double border.
  {*}WPBRD_DOTTED = 5; // \brdrdot    Dotted border.
  {*}WPBRD_DASHED = 6; // \brdrdash   Dashed border.
  {*}WPBRD_HAIRLINE = 7; // \brdrhair Hairline border.
  {*}WPBRD_INSET = 8; // \brdrinset   Inset border.
  {*}WPBRD_DASHEDS = 9; // \brdrdashsm       Dashed border (small).
  {*}WPBRD_DOTDASH = 10; // \brdrdashd       Dot-dashed border.
  {*}WPBRD_DOTDOTDASH = 11; // \brdrdashdd   Dot-dot-dashed border.
  {*}WPBRD_OUTSET = 12; // \brdroutset       Outset border.
```

```
{*}WPBRD_TRIPPLE = 13; // \brdrtriple Triple border.
  {*}WPBRD_THIKTHINS = 14; // \brdrtnthsg    Thick-thin border (small).
  {*}WPBRD_THINTHICKS = 15; // \brdrthtnsg    Thin-thick border (small).
  {*}WPBRD_THINTHICKTHINS = 16; // \brdrtnthtnsg     Thin-thick thin border (small).
  {*}WPBRD_THICKTHIN = 17; // \brdrtnthmg    Thick-thin border (medium).
  {*}WPBRD_THINTHIK = 18; // \brdrthtnmg     Thin-thick border (medium).
  {*}WPBRD_THINTHICKTHIN = 19; // \brdrtnthtnmg      Thin-thick thin border (medium).
  {*}WPBRD_THICKTHINL = 20; // \brdrtnthlg    Thick-thin border (large).
  {*}WPBRD_THINTHICKL = 21; // \brdrthtnlg    Thin-thick border (large).
  {*}WPBRD_THINTHICKTHINL = 22; // \brdrtnthtnlg     Thin-thick-thin border (large).
  {*}WPBRD_WAVY = 23; // \brdrwavy    Wavy border.
  {*}WPBRD_DBLWAVY = 24; // \brdrwavydb      Double wavy border.
  {*}WPBRD_STRIPED = 25; // \brdrdashdotstr  Striped border.
  {*}WPBRD_EMBOSSED = 26; // \brdremboss     Embossed border.  (CSS=ridge)
  {*}WPBRD_ENGRAVE = 27; // \brdrengrave     Engraved border.  (CSS=groove)
  {*}WPBRD_FRAME = 28; // \brdrframe  Border resembles a "Frame."
```

The following properties store the width in twips:

```
  WPAT_BorderWidthL = 66; // Thickness left inner Line
  WPAT_BorderWidthT = 67; // Thickness top inner Line
  WPAT_BorderWidthR = 68; // Thickness right inner Line
  WPAT_BorderWidthB = 69; // Thickness bottom inner Line
  WPAT_BorderWidthDiaTLBR = 70; // Diagonal Line - TopLeft/BottomRight
  WPAT_BorderWidthDiaTRBL = 71; // Diagonal Line - TopRight/BottomLeft
```

These values store the color of the borders:

```
  WPAT_BorderColorL = 72; // Color left inner Line
  WPAT_BorderColorT = 73; // Color top inner Line
  WPAT_BorderColorR = 74; // Color right inner Line
  WPAT_BorderColorB = 75; // Color bottom inner Line
  WPAT_BorderColorDiaTLBR = 76; // Diagonal Line - TopLeft/BottomRight
  WPAT_BorderColorDiaTRBL = 77; // Diagonal Line - TopRight/BottomLeft
```

If the values of all borders are the same, these codes can be used as a shortcut to set the value for a whole group:

```
  // Shortcut - set width and color of ALL lines. - Use Flags to switch on/off
  WPAT_BorderType = 87; // Border Mode ALL LINES AROUND BOX
  WPAT_BorderWidth = 88; // Thickness ALL LINES
  WPAT_BorderColor = 89; // Color ALL LINES
```

This is the most important code. By setting a single bit a border is switched on.

```
  // Border Flags - switch borders on/off
  WPAT_BorderFlags = 90;
  {The following flags switch on certain borders. The type can be defined or inherited}
  {*}WPBRD_DRAW_Left = 1; // Left Border (Default = Single Line = Mode 0) = BlLeft
  {*}WPBRD_DRAW_Top = 2; // Top Border
  {*}WPBRD_DRAW_Right = 4; // Right Border
  {*}WPBRD_DRAW_Bottom = 8; // Bottom Border
  {*}WPBRD_DRAW_All4   = 15; // left + Top+ Right + Bottom
  {*}WPBRD_DRAW_DiagLB = 16; // Cells: Diagonal Top-Left -> Bottom-Right
  {*}WPBRD_DRAW_DiagRB = 32; // Cells: Diagonal Top-Right-> Bottom-Left
  {*}WPBRD_DRAW_Bar = 64; // Border outside (right side of odd-numbered pages, left side
of even-numbered pages)
  {*}WPBRD_DRAW_InsideV = 128; // Rows: Inside Vertical
  {*}WPBRD_DRAW_InsideH = 256; // Rows: Inside Horizontal
  {*}WPBRD_DRAW_Box = 512; // Draw Box around paragraph
  {*}WPBRD_DRAW_Finish = 1024; // Draw bottom border here, even if next paragraph has same
border properties
```

6.5 AppendParCopy

If you need a low level routine to **copy one paragraph and all the included paragraphs** (this can be table rows or table cells) to the destination editor use:

```
var par : TParagraph;
begin
  // Get the reference to current paragraph
  par := WPRichText1.ActiveParagraph;
  // Append it to the active RTFDataBlock
  DestWP.ActiveText.AppendParCopy(par);
  // This moves to the next paragraph! Useful in a loop!
  WPRichText1.ActiveParagraph := par;
  // Format is required sometimes later
  DestWP.DelayedReformat;
end;
```

For better understanding here the source of the AppendParCopy method:

```
function TWPRTFDataBlock.AppendParCopy(var SourcePar: TParagraph; SkipObjects:
TWPTextObjTypes = []): TParagraph;
var toPar : TParagraph;
begin
  if SourcePar = nil then Result := nil else
  begin
    Result := SourcePar.CreateCopy(Self, SkipObjects);
    if Result.ParagraphType = wpIsTableRow then
    begin
      if Empty then
        toPar := CreateTable(nil)
      else begin
        toPar := LastPar;
        if toPar.ParagraphType <> wpIsTable then
          toPar := CreateTable(nil);
      end;
      toPar.AppendChild(Result);
    end else
      AppendPar(Result);
    SourcePar := SourcePar.NextPar;
  end;
end;
```

You can see from this code that this routine tries to add new table rows to a table object which already exists in the text. So instead of moving the complete table you can also copy only selected rows. Since the result value of the AppendParCopy function is the new paragraph you can also do some pro-processing, for example apply certain attributes.

You can also use the SET parameter **SkipObjects** to leave out certain object types, such as [wpobjMergeField] to ignore mail merge fields (the contained text will be copied of course).