

WPTools 7

GUIDE



18.05.2013

Copyright 2005 - 2013 by WPCubed GmbH
created by Julian Ziersch

Table of Contents

Foreword	0
Part I What is WPTools?	1
Part II New features	4
Part III License	10
Part IV Technical Notes	12
Part V Installation	13
Part VI Programming Overview	14
1 Data Structures	14
2 Modify the look and feel of the editor	20
Layoutmodes	21
Hover Effects	25
EditOptions	27
ViewOptions	28
Format Options	29
HideTableBorders	32
3 Toolbar and Action Classes	32
4 Change current writing mode	35
5 Change Page Size and Page margins	35
6 Create an own toolbar	36
7 FastAppendText: Create text with multiple letters	36
Example: FastAppendText	36
Create Sections with FastAppendText	37
Create Lists	37
Save created text	37
8 Get and set the text "as string"	37
9 Header and Footer	38
10 Hyperlinks	42
11 Input TextObjects (i.e. page numbers)	44
12 Insert Text	44
13 Load & Save	45
14 Mail Merge (replace fields with data)	46
Create Field	48
InputMergeField.....	48
InputEditField.....	49
ReplaceTokens.....	49
Low Level.....	50
Customize Field Display	51

Update Field (Insert Text from Database)	52
Start Merge process	52
Event OnMailMergeGetText.....	53
TWPMInsertTextContents	54
procedure Clear.....	54
procedure Abort.....	54
procedure LoadTextFromStream(s: TStream).....	54
procedure LoadTextFromFile(const FileName: string).....	54
function LoadImageFromFile(const FileName: string; w: Integer = 0; h: Integer = 0): Boolean;.....	54
property DataCollection: TWPRTFDataCollection	55
property DataBlock: TWPRTFDataBlock.....	55
property MergeAttr: TWPStoredCharAttrInterface	55
property MergePar: TParagraph	55
property MergeParPos: Integer.....	55
property Obj: TObject.....	56
property CodePage: Integer.....	56
property StringValue: AnsiString.....	56
property WideStringValue: WideString	56
property OldText: string.....	56
property OldFormattedText: string.....	56
property OldIsPlain: Boolean.....	56
property CurrentObject: TWPObject.....	56
property CurrentTxtObject: TWPTTextObj.....	56
property Options: TWPMailMergeContinueOptions	57
property Name: string.....	57
property FieldnamePart: string.....	57
property DatasetnamePart: string.....	57
property Command: string.....	57
property StartInspObject: TWPTTextObj.....	58
property EndInspObject: TWPTTextObj.....	58
property DisplayName: string.....	58
property Format: Integer.....	58
property Modified: Boolean.....	58
property Hyperlink: string.....	58
property HyperlinkName: string.....	58
property IsNull: TWPMInsertTextContentsNull	58
property ReplaceMode: TWPMInsertTextContentsReplaceMode.....	58
Preserve Attributes of text inside a field.....	58
FieldLocate: Modify Field Name or Contents	59
Use TWPMDataProvider.....	60
Hide empty paragraphs	60
Forms/Edit Fields	61
Create ComboBox	63
Working with multiple threads	66
15 Move the Cursor	67
CPChar, CPMoveNext, etc.	68
Search&Replace Text	69
16 Printing - how to set printer properties	72
17 Save and load HTML	74
18 Save to HTML / Load from HTML	74
19 WYSIWYG	74

Part VII Guide

76

1 A) Mini Editor with Delphi 5	78
2 B) Localization - change Language in Dialogs	83
3 C) Work with images	87
Technical Information	87
About Linked Images	88
Example Code	89
Provide a Graphic Popup Menu	93
Images and Mailmerge	95
4 D) How to use the "Default Editor"	95
Using "Old" module	96
Using V7 Module	98
5 E) Ribbon Applications	98
Standard XE3 Ribbons	98
TMS Office 2010	102
Start	102
Populate Edit Menu	104
Populate Style toolbar with Font and Color Selector	105
Populate Table toolbar	107
Make the File menu work	108
Style Scroller	109
6 F) Mini Editor	109
7 G) Low level text creation	112
Create Table from Database	112
Create Table in Code	115
Set Attributes in code (unit WPCreateDemoText)	120
Auto capitalisation	126
8 J) Label Priniting	126
LabelDef	127
Active	129
UnitsInch	129
SheetWidth	129
SheetHeight	130
Vertical	130
Top	130
StartNr	130
Row Count	130
Right	130
Padding	131
Name	131
Left	131
LabelWidth	131
LabelHeight	131
Horizontal	132
ColumnCount	132
Caption	132
Bottom	132
AsText	132
Example Project	133
Initialization	133

After changing sheet size and margins.....	133
Change of column count or row count.....	134
Change of label width and height.....	134
ReadProps and StoreProps.....	134
9 H) Techniques	135
Styles	135
Shared Styles and Style Scroller.....	135
Read/Write CSS.....	137
Add and Assign style.....	140
Work with SPAN Styles.....	142
Numbering.....	143
Tables	145
Set width in inch of a certain column.....	146
Work with % width.....	148
Useful code samples for table handling.....	148
Multiple Editors for the Same Text	149
Create Multi View in Code.....	149
Use TWPRTFStorage.....	150
Simulated MDI (one editor, multiple documents)	151
Watermarks	154
PaintEngine	159
Superprint: Print Booklets and Labels.....	159
Print/Edit elements of TWPRTFDataCollection.....	163
Print Labels (TWPSuperPrint).....	164
Sections	167
Full implementation - shows how to work with sections.....	168
Use utility procedure AppendAsSection.....	170
Use strings in WPTOOLS format with the <new section/> tag.....	170
Append section with individual footer (or header)	170
Syntax Highlighting	171
TWPSynEditHighlight.....	172
TWPCustomSyntax.....	172
TWPXMLSyntax.....	173
TWPXMLRTFSyntax	174
TWFieldSyntax	174
TWFieldBandSyntax.....	174
TextObjects	174
TWTextObj with custom draw event.....	174
Add PDF Export	176
10 M) Appendix	178
MIME Import / Export	178
Reader / Writer.....	178
Demo	179
HTTP Interface	180
Form Setup.....	180
Unit Initialization.....	181
User Action and History Management.....	182
Load Method.....	183
Webbrowser with WPTools.....	184
Source View	185
XML editor mode	186
Work with sub paragraphs	189
Use "External Pages"	191
Create Text Paths	195

Bookmarks	196
11 N) PDF export with wPDF	198
Export to PDF	199
Export using dialog	200
Export using PaintRTFPage()	200
Create edit / memo fields	202
Create a check box field	202
Create embedded data objects	203
12 O) Adding Spellcheck	204
Use WPSpell	205
13 K) WPreporter Addon	207
Comparison to "usual" reporting	208
Calculation in Text and Tables	209
Reporting with WPreporter	211
WPreporter - step by step	214
WPreporter Events	221
Convert text into template	223
Reporter and Bookmarks	225
Token to Template Conversion	226
General Syntax - Fields.....	227
Syntax Highlighting.....	228
Bands	228
Groups	229
Parameters for Fields.....	230
Parameters for Bands and Groups.....	230
Example Template.....	231
14 L) WPPremium Addon	233
Text boxes	234
Footnotes	238
Columns	239

Part VIII Reference 240

1 Reader and Writer	240
2 TParagraph API	242
TParagraph Properties	242
TParagraph Methods	245
AppendParCopy	252
3 TWPRTFDataCursor	253
Drop-Markers	253
Attribute Interfaces	255
4 Manage Style Properties	261
List of 'A' methods	262
ASet/GetBorder	264
5 WPAT_codes	265
Character Attributes	265
Paragraph Attributes	267
Numbering Attributes	268
Border Attributes	269
Table Size and Position	270

Part IX C++Builder Notes	272
1 Trouble Shooting/ C++Builder	273
2 Example: Create dynamic TWPCustomRTFEdit	273
3 Example: Create image object and insert	274
Part X Notes for Upgraders	274
1 ... when upgrading from WPTools 5 or 6	274
2 ... when upgrading from WPTools 4	275
Updated Rendering and Formatting	276
Changed Unit Names	276
Changed Classes	278
Changed Pointers	278
Changed GUI elements	279
New Border Handling	279
Changed Style Handling	281
BackgroundImage	281
Part XI Release Notes - WPTools 7	281
Part XII Release Notes - WPTools 6	283
Part XIII Release Notes - WPTools 5	292
Part XIV PDF Products	311
Index	315

1 What is WPTools?

WPTools is a word processing component for **Borland Delphi and C++Builder**.

It loads and saves RTF files. Its RTF implementation is one of the most complete on the component market (header/footer, paragraph styles, optional footnote support, table header rows).

The "premium" edition also does columns, text boxes and footnotes. It also includes a special XML reader and writer unit.

But WPTools does not only serve to edit text, it is a powerful toolset for mass mailing and, optional, RTF reports.

The package also includes a lean but powerful XML interface (unit WPXMLInt.pas) - it makes it extremely easy to store structured information in memory with integrated support for saving and loading.

The component also comes with a set of glyphs which You may use in Your application when you have licensed WPTools 7.

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 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, it necessary to remove pointer arithmetic completely. Those pointers had been very important in 1996 to enhance performance 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. Since WPTools 5 the engine works with property inheritance, this means if a deeper element does not define a certain property (think of indents, shading and alike) the property is used which was defined at a higher level.

The new RTF-Engine was first used for WPTools 5. WPTools 6 is build upon the stable WPTools 5 kernel, but introduces several interesting new features and optimizes the support for the new unicode enabled compiler Delphi 2009 and Delphi 2010 and XE.

Unlike competing products WPTools 5 implemented separation of data and display from the very beginning. The included style support always made it possible to have a set of styles which is valid for different text at the same time. Further more there always was double page and multi column display.

The new WPTools 7 is mostly based on the architecture of WPTools 5 and 6, however it was necessary to split up some of the pascal units into smaller units. The formatting logic has now been implemented in special units, which have to be linked in when required and can be left out, if not. This helps to make compiled code smaller and also to implement alternative formatting code.

WPTools 7 updates some older components which are still widely used in products, i.e. the TWPToolbar. This unique component makes it possible to create a full featured text editor without writing any code. Now the toolbar is customizable, also at runtime. Without having to spend a lot of time, You can enhance your project by simply linking in the new units.

Special features of Delphi XE3 have also been integrated into WPTools 7.

The migration from WPTools 5 or 6 to 7 has been made as easy as possible - usually you only need to add some new units to the uses clause.

What does WPTools Version 7 do?

a) Word processing (WYSIWYG, page layout view, headers + footers, tab stops, paragraph styles, UNICODE).

This is what it is all about. WPTools is the tool to edit documents, the user can insert multiple header and footer texts and all works in a WYSIWYG manner.

When you compare the word processing features with competing products please check out how natural the tables in WPTools "feel". A table can be wider than the text area, it is possible to select ranges of cells and using the border dialog apply borders to that range as if it was just one cell.

WPTools can also display the pages in multiple columns and supports different page sizes with one document.

b) Special text editing tasks - split screen, collapsible paragraphs, interactive display

WPTools separates the editor logic from the data. This makes it possible to work with the text in memory and have just one or more viewer display this text at different layout modes or zoom levels.

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!

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) Spread sheet

WPTools has incredibly powerful and fast table support. It is fast enough to hold thousands of rows! If you have the "WPReporter Addon" (it is included in the "Bundle") also calculation in tables is possible.

e) Mail merge

It is possible to merge in text, plain or formatted and also images. This is probably the most important feature since WPTools always supported non destructive mailmerge. It is also possible to let the user modify the merged document and read out the changes to save them in the database.

f) 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. The central class is TParagraph. It implements many low level methods to read and write directly into the text.

g) Integrated Label printing.

When activated the logical pages in the document are distributed on virtual labels on a virtual label sheet. This makes it possible to do a mailmerge and then preview or even edit the labels before they are printed. We have not seen such a feature in any other component. WPTools 7b

h) Special HTML support

In general WPTools is optimized for document editing. But for special tasks we have integrated a new and alternative formatting routine to display HTML files.

- i) **MIME support** (based on Synapse).
- j) **Integrated HTML and XML syntax highlighting.**
- k) **Automatic HTTP download** (based on Synapse)
- l) With PREMIUM version only: A special **application server mode** reduces the network traffic required for updating the editor. (The double buffer is then disabled)
- m) **Clipboard control and security options** (copied text may not leave the editor)
- n) **Section support** (pages sizes and header/footer can vary within one document)
- o) With PREMIUM version only: Columns - even with balancing
- p) With PREMIUM version only: Footnote support
- q) With PREMIUM version only: Textboxes (may also have frames and backgrounds)

If you need a text component for Microsoft Access(TM), VisualFox PRO(TM), VisualBasic(TM) or .NET please use our word processing component **TextDynamic**. This component has been especially created for .NET. It offers the same word processing features as WPTools Version 7 does. The concept of the programming API is similar. The .NET version does not use an OCX but an assembly written in C#, but an OCX is also included at no extra charge.

Info: www.textdynamic.com

If you need word processing or text conversion features (RTF to PDF, HTML to PDF, RTF to HTML) on a server (ASP) or with .NET or as ActiveX please check out our product [RTF2PDF / TextDynamic Server](#). It now includes the same powerful API as used by TextDynamic.

2 New features

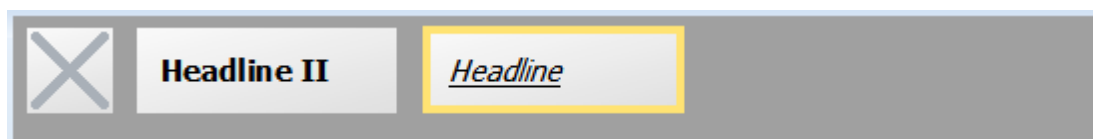
WPTools 7 has been designed to be more accessible than previous version of WPTools. We added several new actions to work with the text. You can easily load and save CSS data. A Style scroller makes it easy to implement a good looking and effective GUI.

The upgrade from WPTools 5 or 6 should be very smooth. In most cases you only need to add a few units to the uses clause (WPRTEPlatform, WPRTEEdit, WPRTEDefsConsts) .

- 1) Many new standard action classes - they already have been set up in the **new unit WPRTEAction7**. All actions have been named with preceding category to make it easier to locate them.
- 2) Method to copy and paste or brush current attributes (CopyToClipboardAttr - action: TWPAAttributesBrush)
- 3) The component TRTFProps makes it easy to share styles between different editors and to create styles at designtime.

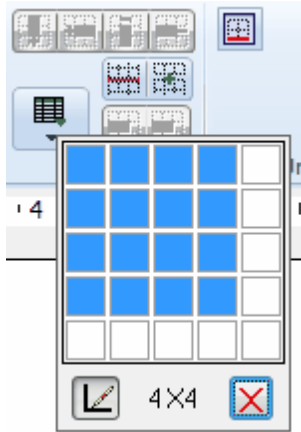


- 4) New Move method (move to hyperlink, table etc)
- 5) New API and handling for sub paragraphs. This makes it easy to hold the data of multiple records in one scrollable editor.
- 6) Images can now be embedded in HTML
- 7) Improved theme support (Delphi XE3). This affects the toolbar, ruler, gutter and editor.
- 8) New hybrid Find and Replace dialog
- 9) New style scrolling component - **TWPStyleScroller**

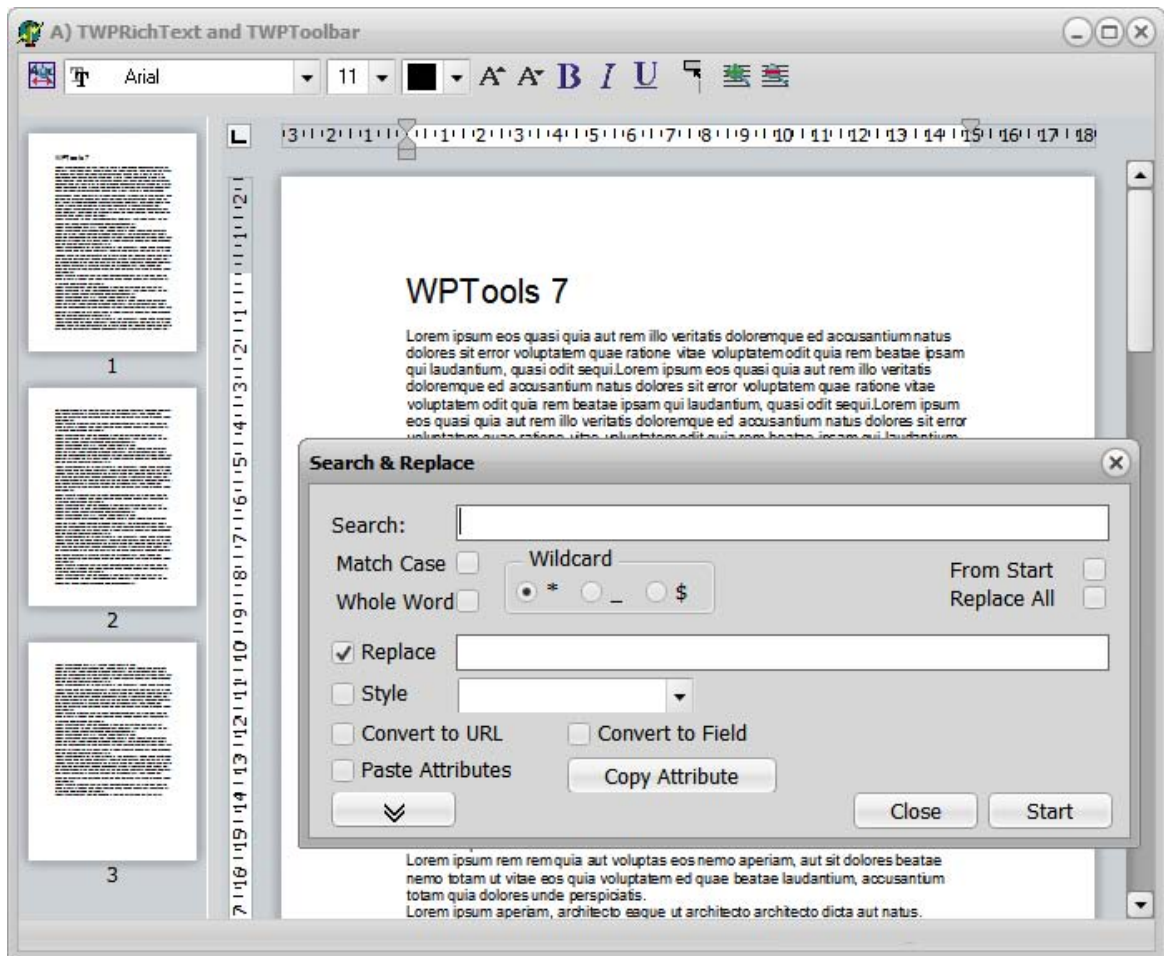


- 10) The units of the RTFEngine have been split up to make it easier to extend it.

- 11) Included glyphs for the standard actions
- 12) Much enhanced API. Method to create demo text: WPLoremIpsum
- 13) Carefully Enhanced GUI (Create Table tool, action behaviour, gutter and ruler)



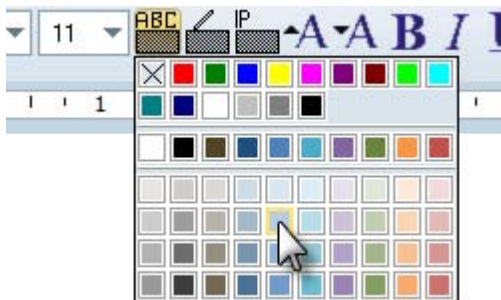
- 14) enhanced label printing
- 15) Improvements to TWPTools, Design and runtime customization
- 16) Improved support for CSS
- 17) New table tools
- 18) You can set a minimum and maximum count of pages. (PageMinCount, PageMaxCount)
- 19) New Find&Replace dialog with more features:



20) function WPToolsRTFTToANSI now can convert unicode characters encoded with RTF \u keyword

21) WPTools 7 supports Delphi XE3 and XE4

22) New Toolbarbuttons and actions to display color drop down: (property WPToolbar.StandardColorDropdowns=false)



Since some customers will upgrade from WPTools Version 5, we list here the most notable new features which have been implemented into WPTools 6. Many of those have been improved for WPTools 7 as well.

1) Application-Server-Mode

WPTools 6 introduces a feature which is called "Application-Server-Mode". **This is only available when you have the PREMIUM version.** This mode is activated when true is assigned to the global boolean variable WPApServerMode. When this mode is activated the editor does not use the double buffered output anymore. While this can cause some flickering the network traffic is reduced when the application runs on an application server, such as Citrix.

Please note, effective with WPTools 6 software which was written for application servers (such as clinic software) may only be distributed when a TEAM or SITE license of WPTools was acquired.

2) [Integrated Label Printing](#)

When activated the integrated label printing shows multiple labels on one virtual sheet of paper. The cursor can move from label to label freely. The user can so edit the labels, add new or delete unwanted labels before the complete sheet is printed. This is a very unique and versatile feature.

3) Additions to the PDF export with wPDF V3

[Create embedded data objects](#)

[Create edit / memo fields](#)

[Create a check box field](#)

Also the creation of PDF tags was enhanced. So now hints to paragraph styles will be also exported.

4) Additional Control over Clipboard Actions

- a) properties to select the format
- b) added security

5) Added [section API](#)

+ WPRichText1.ActiveSection

+ WPRichText1.InputSection

+ TWPPagePropDlg has new method ExecuteEx. Use it to change current page size or Section

WPPagePropDlg1.ExecuteEx(WPRichText1.ActiveSection);

6) [Load over HTTP connections](#) (requires Synapse)

7) [Load and save MIME encoded HTML with embedded images](#) (requires Synapse)

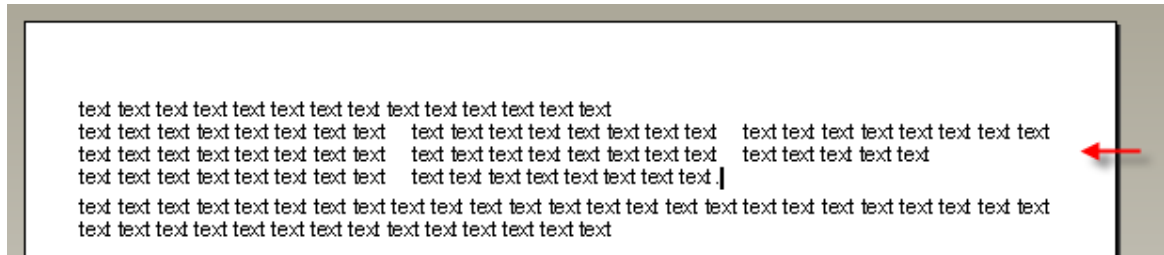
8) [Integrated XML syntax highlighting](#)

9) WPreporter: [Token to Template conversion with syntax highlighting](#)

10) Scroll with middle mouse button

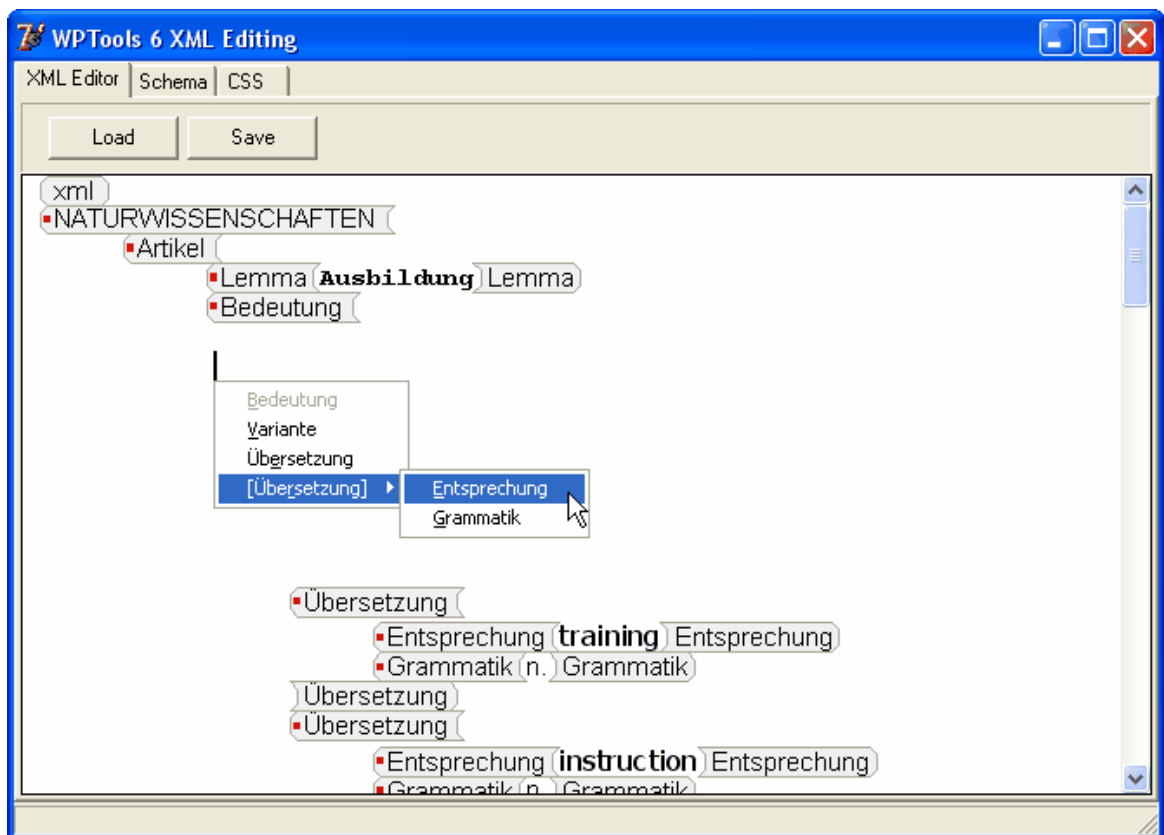
• ↑↓ (when middle mouse button is pressed)

11) Premium Edition: Now Column Balancing is now supported. It is now also possible to draw lines between the columns.



12) draw gradient effect in background of editor

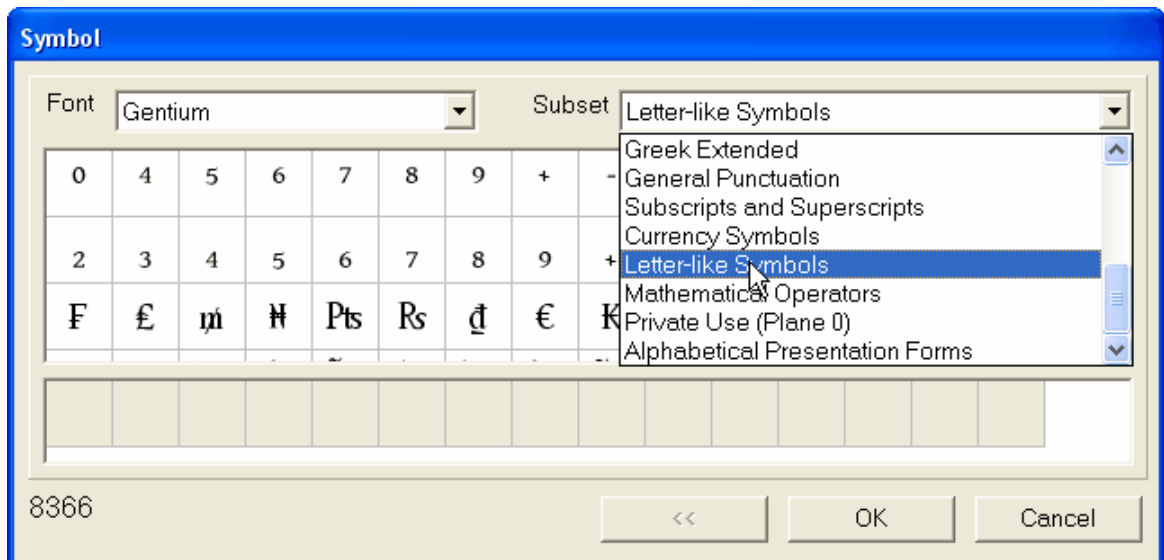
13) [XML editor mode](#)



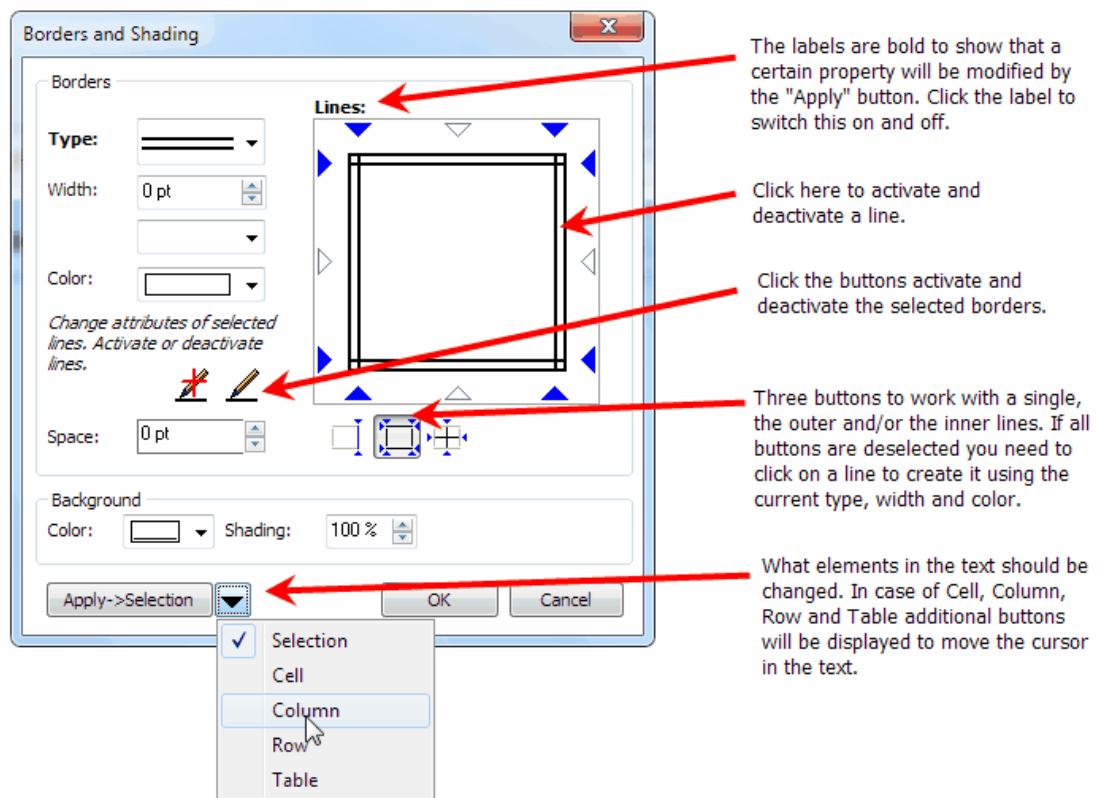
here we use a style sheet to auto format the embedded texts, in this case the style sheet

```
Lemma{font-family:'Courier New';font-weight:bold;color:red}
Entsprechung{font-family:'Tahoma';font-size:13.00pt;}
```

14) new Insert Symbol Dialog



15) a new Border Dialog (will be used by default when compiler symbol NEWBORDER has been activated in WPINC.INC)



3 License

LICENSE AGREEMENT - FULL LICENSE

1. NO ROYALTIES FOR EXE-PROJECTS

WHEN YOU HAVE PURCHASED AND PAID A VALID WPTOOLS DEVELOPING LICENSE, YOU HAVE THE RIGHT TO DISTRIBUTE PROGRAMS (EXE-FILES) YOU HAVE CREATED WITH THE HELP OF WORD PROCESSING TOOLS ROYALTY-FREE. YOU MAY **NOT** DISTRIBUTE MODULES WHICH MAY BE USED BY OTHER (ALSO NOT "INHOUSE") DEVELOPERS (SUCH AS COMMANDLINE TOOLS, VBX, OCX, VCL, DLL, VCL OR ACTIVE-X) WITHOUT WRITTEN PERMISSION. THE LICENSE DOES NOT INCLUDE THE PRODUCTION OF ACTIVE-X MODULES FOR THE USE WITHIN THE INTERNET.

THE SOFTWARE SUPPLIED MAY BE USED BY ONE PERSON ON AS MANY COMPUTER SYSTEMS AS THAT PERSON USES.

GROUP PROGRAMMING PROJECTS MAKING USE OF THIS SOFTWARE MUST PURCHASE A COPY OF THE SOFTWARE FOR EACH MEMBER OF THE GROUP. THIS DOES *ALSO* APPLY WHEN "ONLY ONE" PERSON IN THE GROUP IS DEVELOPING WITH WPTOOLS!

CONTACT WPCubed GmbH (sales@wptools.de) FOR VOLUME DISCOUNTS AND SITE LICENSING AGREEMENTS.

NOTE: THE DISTRIBUTION LICENSE REQUIRES:

IF WPTOOLS IS USED IN A PROJECT WHICH IS DEVELOPED BY A **GROUP OF DEVELOPERS, ALL MEMBERS** OF THIS GROUP MUST HAVE A WPTOOLS DEVELOPMENT LICENSE!

THIS ALSO APPLIES IF ONLY ONE MEMBER OF THE GROUP IS WORKING WITH WPTOOLS (OR WPDF, WPVIEWPDF) DIRECTLY.

IF THIS LICENSING DEMAND HAS NOT BEEN MET, THE DISTRIBUTION OF THE COMPILED PROJECT WILL MEAN A COPYRIGHT INFRINGEMENT. IF DEVELOPERS JOIN THE PROJECT, NEW LICENSES ARE REQUIRED.

(IN CASE ONLY ONE DEVELOPER WORKS WITH WPTOOLS BUT THERE ARE OTHERS IN THE PROJECT WHO WORK WITH OTHER DEVELOPMENT SYSTEMS, THOSE OTHER DEVELOPERS STILL NEED A WPTOOLS LICENSE SINCE THE WORK OF THE FIRST DEVELOPER IS A "MODULE" FOR THE OTHERS.)

PLEASE NOTE, EFFECTIVE WITH WPTOOLS 6 SOFTWARE WHICH WAS WRITTEN FOR APPLICATION SERVERS (SUCH AS CLINIC SOFTWARE) MAY ONLY BE DISTRIBUTED WHEN A TEAM OR SITE LICENSE OF WPTOOLS WAS ACQUIRED.

IF THIS CONDITION IS NOT MET, THE PRODUCT MAY NOT BE DISTRIBUTED.

2. THE LICENSE DOES NOT ALLOW PRODUCTION OF MODULES, DLLS, ActiveX OR COMMAND LINE UTILITIES

UPON REGISTRATION OF THE STANDARD VERSION YOU WILL RECEIVE 70% OF THE SOURCE FILES FOR VERSION 5.x. YOU MAY ALTER THEM BUT YOU MAY NOT DISTRIBUTE THEM TO ANY OTHER PERSON WHO HAS NOT REGISTERED! YOU MAY NOT DISTRIBUTE WPTOOLS-DCU FILES OR DELPHI/BCB DESIGNING PACKAGES EITHER. IT IS NOT ALLOWED TO USE WPTOOLS IN COMMAND LINE UTILITIES, SUCH AS TOOLS WHICH DESIGNED TO ONLY RENDER RTF OR HTML - EXCEPT FOR INHOUSE USE. THE USE IN A GENERAL "VIEWER" APPLICATION WOULD BE AGAINST THIS LICENSE.

IF YOU NEED THE COMPLETE SOURCE PLEASE PURCHASE THE PRO OR PREMIUM VERSION!

3. NO REVERSE ENGINEERING

YOU MAY NOT REVERSE ENGINEER, DECOMPILE, OR DISASSEMBLE THE PRODUCT UNLESS ALLOWED BY APPLICABLE LAW. THE PROVISION OF SOURCE CODE DOES NOT CONSTITUTE A TRANSFER OF ANY LEGAL RIGHTS TO SUCH CODE, AND RESALE OR DISTRIBUTION OF ALL OR ANY PORTION OF ALL SOURCE CODE AND INTELLECTUAL PROPERTY WILL BE PROSECUTED.

THIS COMPONENT IS LICENSED FOR USE WITH DELPHI OR C++ BUILDER WIN32 ONLY - THE SOURCE MAY NOT BE ALTERED TO BUILD .NET TOOLS OR TOOLS FOR OTHER COMPILERS OR OPERATION SYSTEMS.

THE WPTOOLS STANDARD AND STANDARD-PRO VERSION DOES NOT INCLUDE THE RIGHT TO EXTEND IT TO SUPPORT COLUMNS, TEXTBOXES OR THE PRINTING OF FOOTNOTES. THIS IS RESERVED TO THE "PREMIUM" EDITION. THE SAME IS TRUE FOR TEXT BOXES BASED ON OUR LAYER TECHNOLOGY.

THE DISTRIBUTION LICENSE FOR CREATED APPLICATION(S) REQUIRE THIS LICENSE AGREEMENT TO BE RESPECTED!

4. YOU MAY NOT RENT, LEASE, OR LEND THIS SOFTWARE COMPONENT.

ONCE AN APPLICATION WHICH USES THIS LIBRARY WAS DISTRIBUTED, THE LICENSE MUST STAY WITH THE COMPANY WHICH HOLDS THE DISTRIBUTION RIGHT TO THE DISTRIBUTED APPLICATION.

NO TEAM USE OF SINGLE LICENSES:

IT IS NOT ALLOWED TO SELL OR PASS ON DEVELOPING LICENSES FROM ONE DEVELOPER OR COMPANY TO THE NEXT, AFTER THE FIRST COPY OF THE (FIRST) CREATED APPLICATION WAS MADE PUBLIC. ONCE AN APPLICATION WAS MADE PUBLIC, THE LICENSE MUST STAY WITH THE PERSON(S) WHO USED IT IN THIS MOMENT.

5. LIABILITY LIMITATION

THE DOCUMENTATION AND THE VCL ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR SUITABILITY FOR A PARTICULAR PURPOSE. THE USER ASSUMES THE ENTIRE RISK OF ANY DAMAGE CAUSED BY THIS SOFTWARE.

THIS INCLUDES DAMAGE BECAUSE OF INFRINGEMENT OF ANY PATENTS.

IN NO EVENT SHALL JULIAN ZIERSCH OR WPCUBED GMBH BE LIABLE FOR DAMAGE OF ANY KIND, LOSS OF DATA, LOSS OF PROFITS, INTERRUPTION OF BUSINESS OR OTHER PECUNIARY LOSSES ARISING DIRECTLY OR INDIRECTLY FROM THE USE OF THE PROGRAM. ANY LIABILITY OF THE SELLER WILL BE EXCLUSIVELY LIMITED TO REPLACEMENT OF THE PRODUCT OR REFUND OF PURCHASE PRICE.

GOOD DATA PROCESSING PROCEDURE DICTATES THAT ALL PROGRAMS BE THOROUGHLY TESTED WITH NON CRITICAL DATA BEFORE THEY CAN BE RELIED UPON.

4. COPYRIGHT MESSAGE REQUIRED

IF YOUR PROGRAM HAS AN "ABOUT BOX" THE FOLLWING CREDIT SHOULD BE DISPLAYED IN IT: "WPTools (C) Julian Ziersch" or "WPTools (C) WPCubed GmbH"

4 Technical Notes

WPTools Version 7 supports Delphi 5,6,7, 2006 (Win 32) and Delphi 2007. It also supports Borland C++ Builder 5 and 6 and C++Builder 2006. It also supports Delphi 2009, Delphi 2010, XE, XE2, XE3 and XE4.

WPTools 7 PRO and PREMIUM also supports Delphi XE2, XE3 and XE4 64bit.

Firemonkey is not supported by this edition of WPTools. It has not been decided if there will be a WPTools FMX edition. (POLL)

Delphi 3 and 4 are not supported since the code requires modern language features, such as method overloading.

If you need a text component for Microsoft Access(TM), VisualFox PRO(TM), VisualBasic(TM) or .NET please use our new word processing component **TextDynamic**. Info: www.textdynamic.com.

For ASP and ASP.NET we have **RTF2PDF / TextDynamic Server**. This component includes most of the powerful interface methods TextDynamic has.

Please check out our demo webserver at <http://www.rtf-net.com/>

The code uses pointers only in very few functions - mainly to optimize the performance. The reader and writer classes do not use static buffers.

The use of global variables has been limited. The most important ones are stored in the TWPToolsEnviroment object. By creating several instances of this class it is possible to create a threadsave application. (See demo Mailmerge\ThreadSave)

5 Installation

To reduce the size of the setup exe we have not included compiled packages. We could include BPLs, but this would increase the download size so much, that it would cause problems. We also tried to make the concept of the sources as simple as possible, by avoiding lots of different paths which would make it hard to recreate a working compiler configuration after a change of the machine.

If you also have WPSpell or wPDF, simply decompress those (run installer) and then open the file WPTools\Dxx\WPINC.INC and activate the respective \$DEFINE for the product to be included. Only then do a build with the wptools package.

To compile the packages please open the *.DPK file in the directory WPTools\Dxxx (xxx depending on the version of Delphi) and click on 'install'. (The option "install" is in the package popup dialog or, in later Delphi versions a sub menu in the context menu of the project management)

Please check if the library path under "Enviroment Options" list the directory of the WPTools units (WPTools\Dxxx,...).

If there is a previous version of WPTools on your computer its package must be removed from the Delphi package list and the path must be removed from Delphi's search path.

Troubleshooting:

If the IDE reports an error message about a missing WPTools unit, please make sure the WPTools\Dxxx directory is listed under library paths - xxx = delphi version. (The setup runs as a different user and does not have access to the current user registry items)

If the XE IDE reports an error message about an missing VCL unit, please make sure the unit namespaces (Project options) list "Vcl;System" and others.

The demo and the standard editions have been compiled to work with the registered versions of Delphi. They will usually not work with trial editions of Delphi.

If the WPTools does not allow editing, loading or saving of text, it means the demo is expired.

Applications which have been compiled with the Demo edition will only work on a machine where the WPTools demo was installed..

If you have questions, please do not hesitate to ask by mail to support@wptools.de

6 Programming Overview

6.1 Data Structures

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

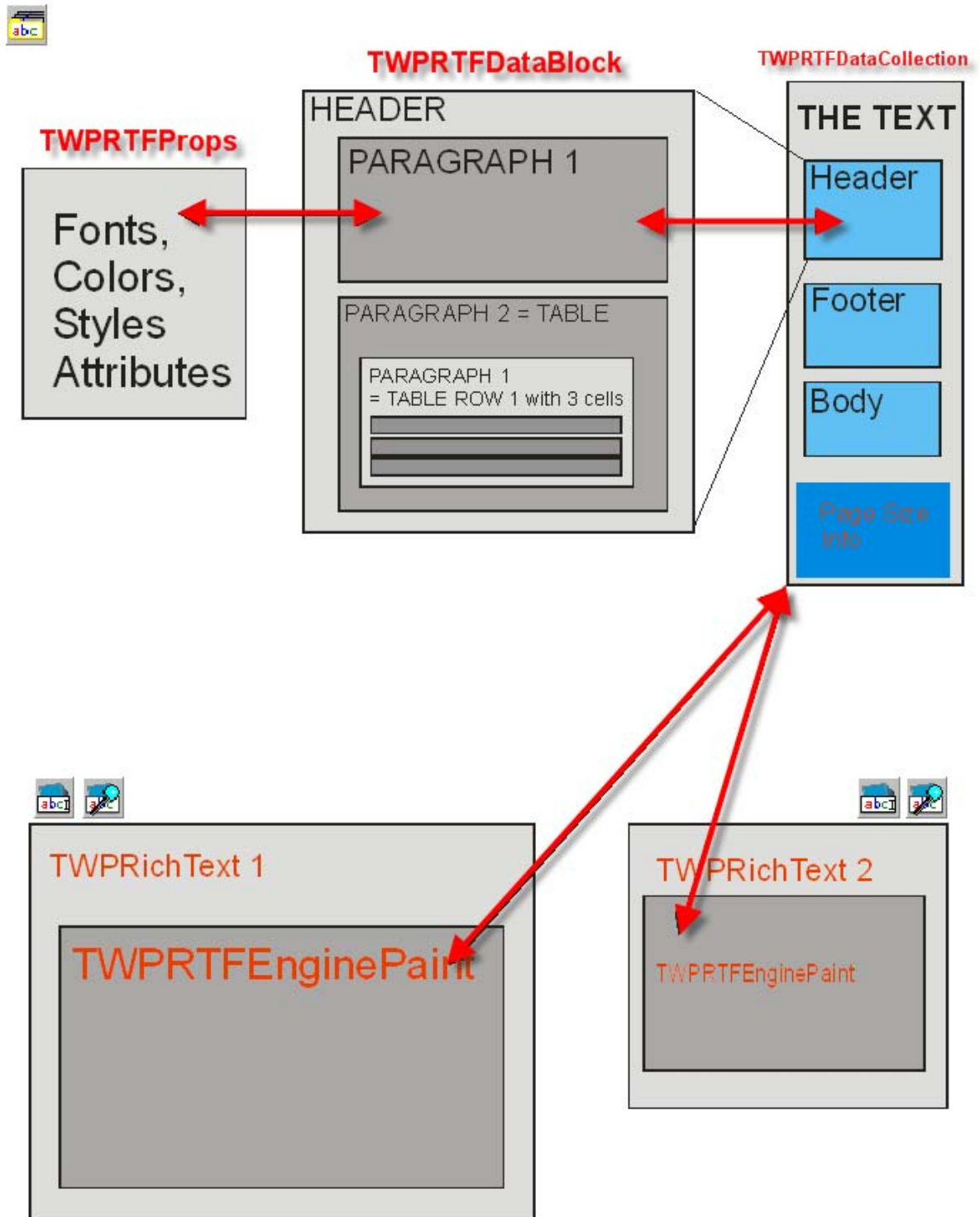
Please read this carefully! This knowledge is necessary for you to understand WPTools Version 7.

Note: the separation between edit/display and data-objects make some elegant solutions possible - ie. create MDI editor without using MDI windows! (See demo [DynAssign](#))

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

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** (see: [Multiple Editors for the Same Text](#))
- b) RTF properties, such as paragraph or number styles, are stored in the **TWPRTFProps** object. (see: [Share Styles between TWPRichText](#))



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

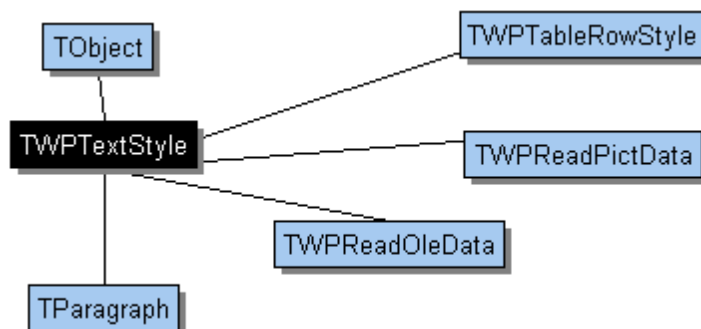
all editors attached. The cursor object also controls text selection and changing properties of the selected text (SelectedTextAttr : **TWPSelectedTextAttrInterface**) or the current writing mode (CurrentCharAttr : **TWPCursorCharAttrInterface**). It also contains the CPAttr (**TWPTAttrEmulator**) interface which changes the attribute at the cursor position. (Please see last inheritance chart and "Character Attributes" below)

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.

To copy the first row of a table after the current row you can use this simple code:

```
current_row.NextPar := current_table.RowFirst.Duplicate(true, true);
```

In this code the first row is duplicated and inserted into the chain of rows by assigning it to the **NextPar** property. **Duplicate()** needs two parameters, the first enables the copying of the text (otherwise only the properties are copied) the second enables the copying of the children, in case of a table row this are the cells.

Paragraph Attributes:

WPTools Version 7 supports many different paragraph, table and border attributes. These attributes are identified by a code, the [WPAT_code](#). (The constants all start with WPAT_). **It is important to remember that not all property ids make sense in all TParagraph objects**, for example a table row cannot use the column width property. Some properties override each other (WPAT_ColWidth override WPAT_ColWidth_PC) and some are reserved for future versions. To read a attribute you can use the method TParagraph.**AGet**(code, value). 'Value' is passed as **var** parameter (by reference) and is only modified if that property was defined by this TParagraph or TWPTTextStyle. In case the property was defined the function AGet returns true, otherwise false. Alternatively you can use the function **AGetDef**(code, default_value). Here the value of the property is returned if defined, otherwise the provided default value. There are more 'A' methods, to delete a property (ADel), to read the properties of the paragraph or the style it uses (AGetInherited), to read and set a color value (AGetColor, ASetColor). Please see the TParagraph reference in the HLP file.

Important: The method TWPTTextStyle.AGet(WPAT_code : Integer; var Value : Integer) is a functions which returns a boolean value. The return code is false if the property with the id WPAT_code was not defined. In this case the variable "Value" will not be modified!

Please make sure you initialized the variable Value!

Tabstops:

Tabstops are not stored as WPAT_ properties. They are accessed through several methods, such as TabstopAdd, TabstopMove or TabstopGet. (See reference)

Character Attributes:

As stated character attributes are stored in just one double byte value. You may ask, 'How can this work?' Particularly since WPTools Version 7 supports 15 different character attributes with multiple settings possible for each of these.

WPTools Version 7 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, such as

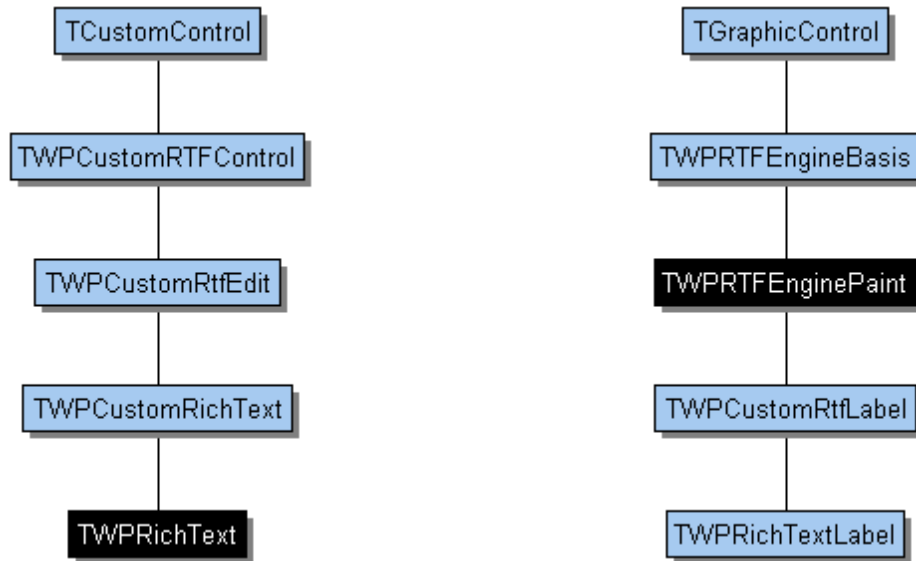
TWPAbstractCharAttrInterface, make it easy to create and use the index values.

```
WPRichText1.AttrHelper.Clear;  
WPRichText1.AttrHelper.SetFontName('Courier New');  
WPRichText1.AttrHelper.SetColor(clGreen);  
WPRichText1.ActiveParagraph.SetText(  
    'Some green text',  
    WPRichText1.AttrHelper.CharAttr);  
WPRichText1.DelayedReformat;
```

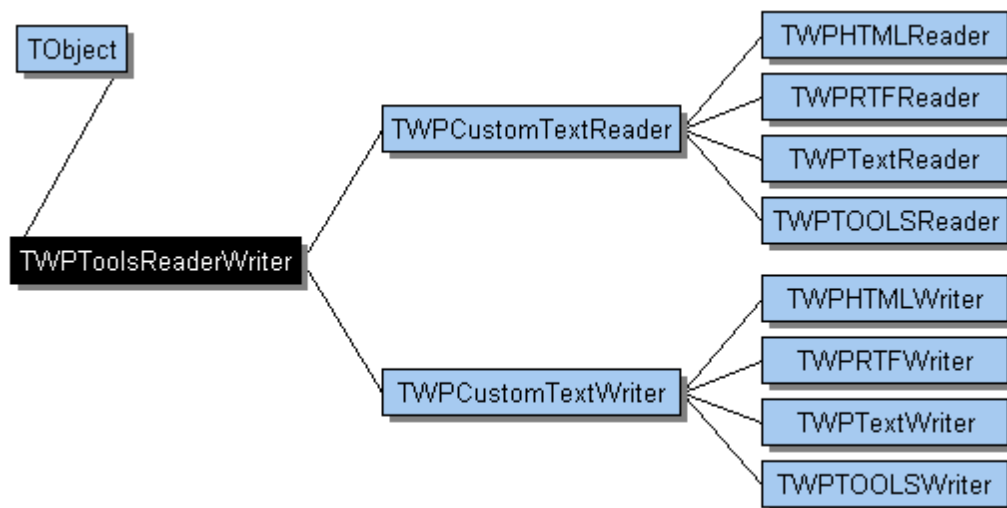
Explanation: AttrHelper is an object of class TWPStoredCharAttrInterface. It calculates "CharAttr" index values. 'Clear' will delete all attributes - the CharAttr index will be 0. SetFontName and SetColor are used to define new character attribute. Reading the property CharAttr (inside the call to the TParagraph method SetText) will create a new CharAttr index which is used for the text. The calculated CharAttr can be used at different places for text which should look the same. It will become invalid when the document is cleared by TWPRichText.Clear. Read [more...](#)

Inheritance Charts:

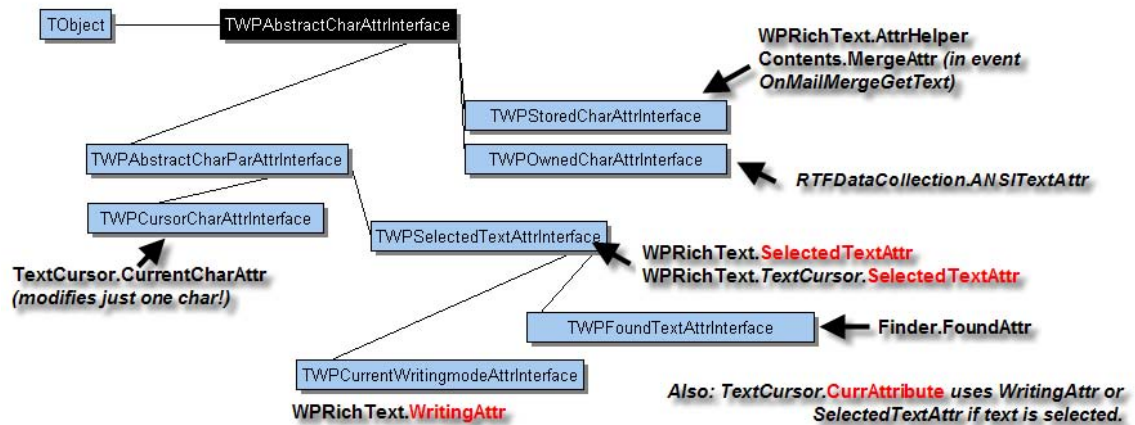
TWPRichText, the RTF memo and TWPRTFEnginePaint, the RTF Engine (used by the TWPRichText as object 'Memo', the TWPRichTextLabel inherits from it)



Please note that we now use format strings to pass properties to the **reader and writer classes**. Example: `WPRichText1.AsANSISString('RTF-onlybody')` creates a string in RTF format which contains only the body text.



To update TParagraph and TWPTextStyle objects you will have to use the "A" methods (ASet, AGet) - the **interface classes** are only used to either change the current writing mode or the attributes of multiple paragraphs and characters (such as selected text).



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.

6.2 Modify the look and feel of the editor

Please use the properties ViewOptions, ViewOptionsEx, EditOptions and EditOptionsEx of the TWPRichText.

The TWPToolbar and TWPRuler both have a property DrawOptions.

Example:

```

WPToolbar1.DrawOptions := [ wptDrawPageShade ];
WPToolbar1.FlatButtons := true;
WPToolbar1.MarginBottom := 8;
WPToolbar1.BevelLines := [ wplBottomShade ];

```

```

WPRuler1.DrawOptions := WPRuler1.DrawOptions + [wpDrawThemedBackground,w
WPVerRuler1.DrawOptions := WPVerRuler1.DrawOptions + [wpDrawFramelines
WPRuler1.DrawOptions := [wpDrawPageShade];
WPVerRuler1.DrawOptions := [wpDrawPageShade];
WPRichText1.ViewOptionsEx := WPRichText1.ViewOptionsEx + [wpPaintPageSha
WPPreview1.ViewOptionsEx := WPPreview1.ViewOptionsEx + [ wpPaintPageSha

```

When the DrawOption wpDrawPageShade and ViewOptionEx wpPaintPageShade is used, a bitmap is used to render the shaded borders of the page. This is done by the utility function **WPDDrawRectWithBitmap**.

To use a different bitmap instead of the default one, you can assign it to the global variable WPDDrawRectWithBitmap_bitmap.

Example: `WPDDrawRectWithBitmap_bitmap := MetroStyleImage.Picture.Bitmap;`

The bitmap should show a shaded square. To display a also a shade in focussed state, two squares must be rendered side by side. A good size for one saque is 66*66 pixel since internally it is divided into 9 parts.

Example bitmaps for WPDDrawRectWithBitmap:



The properties `FormatOptions` and `FormatOptionsEx` change how the formatting routine works, they have a significant influence on how the text appears.

Please also see the [Layout Catagory](#).

To change the language (for localization, translation) please see demo "[Lozalisation](#)".

6.2.1 Layoutmodes

The different layout modes are probably one of the most exciting feature in WPTools Version 7. 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.

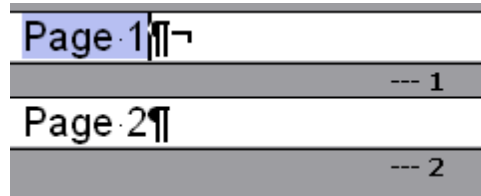
wplayShowManualPageBreaks - like `wplayNormal` but draws a dashed line for page

breaks

wplayPageGap - display the text similar to wplayNormal with a bar in 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.

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

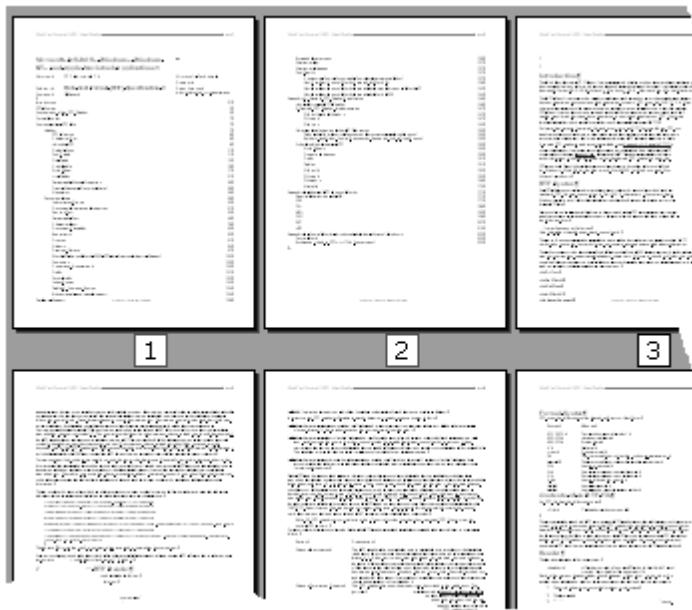
wplayLayout - show PageLayout but do not paint header and footer.

wplayFullLayout - show PageLayout with header and footer.

wpDualPageView - display 2 pages side by side. A similar effect can be achieved 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 outside 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 the selection 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 shade 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

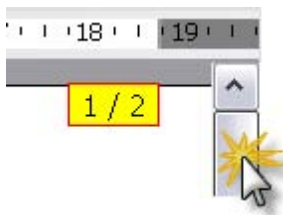
wpCenterPaintPages - center the page horizontally in the window. This is useful for preview windows.

wpDrawPageMarginLines - draw dotted lines at the page margins.

wpDontDrawSectionMarker - do not draw the arrow which shows where a new section starts.

wpDrawHeaderFooterLines - draw gray lines around header/footer areas.

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.



wpDontDisplayScrollPageHint - Do not display this default page number hint when pressing the scrollbar handle. The hint can be modified by changing the global string variable WPPageHintStr. The default is ' %d / %d '.

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.

new: wpAutoZoomHorizontalTiles: You can use wpAutoZoomHorizontalTiles to display the pages in a horizontal scrolling window.

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

6.2.2 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 TWPTTextObj 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 OnHintEventsActive 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 "HotStylelsActive" the property "OnHintEventsActive" 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 HotStylelsActive set to true in the respective TCharacterAttr property.

This code can be used to show a hint window with information about merged text. HotStylelsActive 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.

6.2.3 EditOptions

The properties EditOptions and EditOptionsEx control how the editor works. They are also used to disable certain actions, such as table resizing.

```

TWPEditOptions = set of
  (wpTableResizing, // Move left right indent }
  wpTableOneCellResizing, // always only one cell, can be switched with CTRL key
  wpTableColumnResizing, // change column width
  wpTableRowResizing, // Change height of row. Also see EditOptionEx:
wpTableRowResizingWithCTRL
  wpClearAttrOnStyleChange, //ON: clear the redundant properties when the style name is
changed.
  wpNoAutoWordSelection, // don't select complete words (like Word)
  wpObjectMoving, // move images (ObjType=wpobjImage)
  wpObjectResizingWidth, // the width of objects can be changed
  wpObjectResizingHeight, // the height of objects can be changed
  wpObjectResizingKeepRatio, // the width/height of objects can be changed
  wpObjectSelecting, // objects can be selected
  wpObjectDeletion, // objects can be deleted (only used for TWPObject)
  wpNoAutoScroll, // Switch off the new Auto Scroll Feature
  // obsolete! wpFieldObjectsAsGraphicObjects, { work with TWPOFieldObject as if they were
TWPOGraphics }
  wpSpreadsheetCursorMovement, { Cursor up/down in Rows }
  wpAutoInsertRow, { wpAutoInsertRow, TAB in last cell. Must be combined with
wpSpreadsheetCursorMovement }
  wpNoEditOutsideTable, { to simulate spreadsheet - V5 ok }
  wpBreakTableOnReturnInRow, { V5: instead of inserting a row break up the table }
  wpActivateUndo, { activate UNDO }
  wpActivateUndoHotkey, { activate ALT + Backspace - requires wpAllowUndo set too }
  wpActivateRedo, { makes backup of complete text ! }
  wpActivateRedoHotkey, { makes backup of complete text ! }
  wpAlwaysInsert, { don't switch to overwrite }
  // wpDeactivateCharsetSwitching,
  wpMoveCPOnPageUpDown, { Move Cursor on Page up or Down code - V5 ok }
  wpAutoDetectHyperlinks, // Create a hyperlink after 'www.' was typed
  wpNoHorzScrolling, //V5=ok
  wpNoVertScrolling, //V5=ok
  // wpNoAutomaticHangingBulletsAndNumbers, - see FormatOption!
  // wpMDIDragAndDrop,
  // wpDontDeleteExtraSpace,
  // wpUseHyphenation, -> Moved to FormatOptions!
  wpDontSelectCompleteField, // Selections will always wrap the complete merge field
  wpSelectCompleteFieldAlsoWhenInside, // When selection inside of field also complete field is
selected
  wpStreamUndoOperation, // saves also additional info like bands, objects ..
  //is default: wpToolBarDisableDifferentFontsInSelections, // Sets Size,Color, Font to blank is not
= in a selection
  wpTabToEditFields, // not used.
  // wpAllowEditHeaderFooter
  wpSelectPageOnDbClick, //V5=ok
  // Used by TWPRichText only:
  wpAllowCreateTableInTable // - the create table button allows nested tables
  // Don't allow selection at all
);

```

```

TWPEditOptionsEx = set of
(
    wpDisableSelection, // The user cannot select text (see ViewOptions to hide selection)
    wpDisableCaret, // The caret (insertion point marker at cursor position) is not displayed
    wpDisableGetFocus, // The editor will never receive the focus
    wpDisableEditOfNonBodyDataBlocks, // in Pagelayout mode other DataBlocks (header, footer)
cannot
    // be selected for editing with a click of the mouse
    wpAllowCursorInRow, // the cursor can be placed in row end marker to create a new row with
return
    wpTextObjectMoving, // move text objects (ObjType=wpobjTextObj)
    wpTextObjectSelecting, // By default allow selection of text objects
    wpNoAutoWordStartEndFieldSelection, // Normally a complete field is selected
    // when the cursor is moved over the start or end of a mail merge field.
    // unless wpNoAutoWordSelection is used. Also see EditOption:
wpSelectCompleteFieldAlsoWhenInside
    // Note: wpNoAutoWordStartEndFieldSelection only applies to selection with mouse!
    wpDisableAutoCharsetSelection, // If true the charset is not retrieved by
    // checking the current keyboard layout when the user types
    wpIgnoreSingleCellSelection, // No cell selection by pointing in bottom left corner
    wpTABMovesToNextEditField, // used with forms
    wpDbClickCreateHeaderFooter, // Double click in Margin creates header/footer for all pages
    // Use 'OnClickCreateHeaderFooter' event to modify 'range'
    wpRepaintOnFieldMove, // When the cursor moves to a different field
    // the form is repainted. This is important if you use code to
    // highlight the current field. (using event: OnGetAttributeColor)
    wpKeepCellsWhenCombiningCells, // use the HTML way to combine cells horizontally
    wpAllowSplitOfCombinedCellsOnly, // use the HTML way to split cells, dont allow split if not
combined
    wpDontClearStylesInNew, // If defined Action 'New' will not clear the defined styles
    wpDontResetPagesizeInNew, // If defined clearing the text will not set up the default page size
    wpSetDefaultAttrInNew, // If defined "New" will preset the writing attributes to the default
    wpAllowDrawDropBetweenTextBlocks, // Allow drag & drop between header and body
    wpDisableFastInitOnTyping, // Disable the improved typing performance in large paragraphs
    wpDontTriggerPopUpInContextEvent, // Changes the time the event OnMouseDownWord is
triggered (old behaviour)
    wpAlwaysColWidthPC, // Changine column width sets percentage colwidth in cells
    wpDisableXPosLineUpDown, // Disable the code which tries to keep cursor at same X position in
Line up/down
    wpDontInitSelTextAttrWithDefaultFont, // SelTextAttr will not report the default attr for undefined
    wpDontSelectCellOnSpreadsheetMovement, // when tabbing through a table dont select the cell
    wpScrollLineWise, // When using line up&down do not move by 1/3 screen but only by the height
of a line
    wpZoomWithMouseWheel, // Press Ctrl + use Mouse Wheel to zoom in and out
    wpTableRowResizingWithCTRL, // Resize rows when also CTRL is pressed
    wpDontUseNumberIndents // when using Inc/Dec Outline Level ignore the indents defined in
number styles
    wpAutoCapitalize // WPTools 7: Auto capitalize sentence starts
);

```

6.2.4 ViewOptions

The properties ViewOptions and ViewOptionsEx (added in WPTools 7) control how the text is displayed.

6.2.5 Format Options

WPTools 7 uses the properties

FormatOptions
FormatOptionsEx
and FormatOptionsEx2

to change, how the text is formatted. If changed for one RTF-Engine all other RTF-Engines which use the same RTFData object will be affected, too.

You need to call ReformatAll(true,true) to update the text.

```
TWPFormatOptions = set of (
    // This flags change the display of tables
    wpDisableAutoSizeTables, // - we suggest to enable this flag. Otherwise
    wpNoMinimumCellPadding, // Do not add a one pixel padding to all cells
    wpfDontBreakTables, // do not break tables at all. Also see wpKeepToge
    wpfDontBreakTableRows, // do not break table rows
    wpDontClipCells, // do not clip cells if absolute row heights are used
    wpfIgnoreMinimumRowheight, // Do not use the row height property
    wpfIgnoreMaximumRowheight, // Do not use the row maximum height property
    wpTableRowIndicator, // must be combined with EditOptionsEx : AllowCur
    // -----
    // This flags control the layout
    wpfIgnoreKeep, // The WPAT_ParKeep property is ignored (do not break p
    wpfIgnoreKeepN, // The WPAT_ParKeepN property is ignored (do not break
    wpfKeepOutlineWithNext, // Text which uses the WPAT_ParIsOutline property
    wpfAvoidWidows, // Avoid single lines on old page
    wpfAvoidOrphans, // Avoid single lines on new page
    wpfCenterOnPageVert, // Center text on all pages
    wpfHangingIndentWithTab, // V5: first tab in paragraph jumps to indent
    // the left indent will be handled as first tabstop not only if the tab
    // first character but also if the text before the tab fits into the i
    wpfDontTabToIndentFirst, // V5: The opposite to wpfHangingIndentWithTab
    wpJustifySoftLinebreaks, // Justify \n
    wpJustifyHardLinebreaks, // Justify \r
    wpUseHyphenation, // Use soft hyphens in the text (inserted with Ctrl
    wpFooterMinimumDistanceToText, // The footer texts start after the bo
    // -----
    // This flags control the display of codes.
    // IMPORTANT: You need to call ReformatAll(true,true) after you have c
    wpShowBookmarkCodes, // display bookmark tags
    wpShowHyperlinkCodes, // display hyperlinks tags
    wpShowSPANCodes, // display SPAN tags
    wpShowInvisibleText, // show text which would be otherwise hidden
    // -----
    // Experimental flags
    wpfHideEmptyParElements, // reserved: Can be used when editing HTML f
    wpfXMLOutlineMode, // Debugging mode: Show paragraph tree similar XML
    wpWriteRightToLeft, // activate RTL writing
```

```

wpAutoWriteRightToLeft, // reserved for future
// -----
// Troubleshooting flags
wpUseAbsoluteFontHeight, // Calculate the height of the text using the
wpfAlwaysFormatWithScreenRes, // .. even if RM600 is defined in WPCtr
wpDisableSpeedReformat, // format only the current page and the next p
wpDontAdjustFloatingImagePosition // do not adjust image position to h
// note that it is not possible to click on an image which is outside
);

TWPFFormatOptionsEx = set of (
// -----
wpDontAddExternalFontLeading, // When measuring the font height don't
// value defined for a font: Metrics.tmExternalLeading. This improves
// Alternatively set global variable WPDoNotAddExternalFontLeading :=
// activate this mode for all editors!
// -----
// Layout flags
wpfKeepTablesInTextArea, // If defined avoid that table go into right
wpKeepTogetherAlwaysNewPage, // if wpfDontBreakTables is used a table
wpKeepTogetherAdjacentTables, // When tables are not seperated by para
wpDontUseTablePadding, // Do not read the padding of cells from table
wpfIgnoreVertAlignment, // Switches off the vertical alignment
wpfKeepNUsesParImages, // When using KeepN paragraph aligned images w
// kept on same page as their anchor paragraph. (changes the value ca
// Note: Images which use the wrap mode wpwrUseWholeLine will always b
wpDontIgnoreSpacebeforeOnTopOfPage, // switch "Word" mode off
// -----
// Limit the use of inherited attributes
wpDontUseRowPadding, // Do not read the padding of cells from table ro
wpDontUseBorderPadding, // Do not use the border width to calculate pa
wpDontInheritCellAttr, // Dont inherit cell attributes from table row
wpDontUseSPANStyles, // Do not use the stalyes of hyperlink and SPAN c
// -----
// modify the way the editor works
wpAlwaysContinueBorderAfterCR, // Paragraph.SplitAt (= ENTER key in ec
// the new paragraph if the split position is at the end
wpNoDefaultParStyles, // Disable default paragraph style handling (So
// it yoursel in the OnInitializePar event)!
wpfAutoRestartSimpleNumbering, // Reset a numbering level after a line
wpfNestedNumberingInTableCells, // Each table cell uses its individual
// this only affects numbering which does not use outlines
wpfSimpleNumberingPriorityOverOutlines, // When outlines (numberlevel:
// numbered paragraphs the simple numbering will continue AFTER the ou
wpfNoAutoDecTabInTable, // Do not automatically align to only decimal
wpfNoTableHeaderRows, // Ignore table header
wpfNoTableFooterRows, // Ignore table footer
wpfDisableJustifiedText, // Format Justified text as if it was Left A
// -----
// Experimental Flags

```

```

wpAllow_WPAT_NoWrap, // If active don't wrap par marked with WPAT_NoWrap
wpfMixRTLText, // If a paragraph which uses the paprRightToLeft attribute
// western text is not reordered
wpfStoreWPObjectsInRTFDataProps, // unless this flag is used the TWPObjects
// be stored in the TWPRTFDataCollection and not in the TWPRTFDataProperties
// -----
// Activate User CharStyle
wpCharEffectIsUserAttribute, // Currently the character attribute WPAT_Change
// It can be used to store custom information. If the flag wpCharEffectIsUserAttribute
// the RTF engine will ignore this property.
wpfIgnoreTrailingEmptyParAtFooter, // Empty paragraphs are ignored at footer
wpfDontHideParAboveNestedTable, // allow empty paragraph before a nested table
wpfDontCombineDifferentStylePars, // DeleteParagraphEnd will not merge paragraphs
// Can be selectively set for certain paragraphs using WPAT_ParFlags
wpfNumberingControlledByStyles, // Only supported by WPT format!
wpfSectionsRestartFootnoteNumbers,
wpfDontIgnoreKeepNInTable, // unless this flag is set the KeepN in table
wpfDontIgnoreEmptyHeader
);

//:: New HTML formatting routine added to WPTools V6
TWPWebPageFormat = set of (
    wpFormatAsWebpage,
    wpNoMinimumWidth,
    wpNoMinimumHeight); // + formats

TWPFormatOptionsEx2 = set of
(
    wpfSectionsRestartPageNumbers
    // All pages restart their numbering
    , wpfUseKerning
    // initialize the text and also adjust kerning according to font information
    , wpfCodeObjectAsXMLTags // visualize the code objects as XML tags
    , wpDisableBalancedColumns // dont use the new column balancing
    , wpfCodeObjectCheckParStyles
    // code objects format the contained text according to styles with the
    , wpfNewKeepNSupport
    // (Default =on) better Keep N Support which also handles tables. Disable
    , wpfHideParagraphWithHiddenText
    // (Default =off) If a paragraph contains nothing but hidden text, it
    // this is also true, if it is empty and its default attribute contains
    , wpAllowFullWidthImagesInHeaderFooter
    // allow images in header and footer which are as wide as the page
    // even if they are not full page size
    , wpMarginMirrorBookPrint // makes V6 work like V5
    , wpIgnoreAllTabsInPageMargin
    // WPTools 6 will ignore all tabs after the right margin (like V5 does)
    , wpAlwaysContinueShadingAfterCR
    // If true, an enter at first position will not remove the shading of
    // also see wpAlwaysContinueBorderAfterCR in FormatOptionsEx

```

```
, wpNewReformat // Use new 2012 Reformat method. Requires Unit WPRTEF
, wpUseDIVTree // Requires WPTools 7 PRO or Premium
, wpfShowProtectionObjects
, wpfLabelAllowSoftPagebreak
, wpfIgnoreHardPagebreaks
, wpfIgnoreSoftPagebreaks // text which does not fit on a page will be
);
```

6.2.6 HideTableBorders

This property has been added to WPTools 7 to allow hiding of table borders.

Possible values are:

```
wpDontHide,
wpHideOnScreen,
wpHideOnPrinter,
wpHideAlways
```

6.3 Toolbar and Action Classes

WPTools contains a complete set of component to quickly build a user interface.

You can use the ready-to-use TWPToolbar to create an editor within 5 seconds, or you can use the action classes and your own choice of tool buttons to let the user change the attributes of the text.

a) Toolbar Components

The tool-bar components (TWPToolBar and TWPToolPanel) can be attached to any TWPRichText component using its property *WPToolBar*. If you have more than one use the property *WPToolBar.NextToolBar* to connect to the next toolbar in the chain. When the TWPToolBar is used on a **MDI child window** you need to set the global variable *WPIsMDIApp := true*;



TWPToolBar

This components makes it easy to start a new program which uses a TWPRichText component.

You only have to activate some flags in the object inspector and the TWPToolBar will display action and status buttons and some combo boxes to select the font or color of the text in the TWPRichText control.



new: WPTools 7 implements several improvements to the TWPToolbar:

- a) Several new buttons, i.e. Grow/Shrink Size
- b) property DrawOptions allows adaptation of background color to ruler and editor
- c) property ConfigString - it holds the selection and order of the buttons and combos which can be set and arrange in property editor (double click on toolbar!)

When ConfigString is not empty, the "old" sel_.... properties are inactive.

d) property ActionList. The included actions will also be available in designtime and runtime customization

e) customization at runtime is possible: WPToolbarConfigure(WPToolbar1, Self, 'Configure Toolbar');

f) property WPImageList to provide images for different button heights and as PNG data. (See demo in Demos\A)_Mini_Delphi5_Editor\XE3)



TWPToolPanel

Contrary to the toolbar, which creates and positions the buttons automatically, with the tool-panel component you have to drop the buttons in place. This gives you more freedom in the design of the user interface.



TWPCoMboBox

This components can be used together with a TWPToolPanel or TWPToolCtrl. It can control the font, font size or colors of the text - to select the mode use the property ComboBoxStyle.

You can also use this control on a different toolbar or ribbon, if you create a TWPToolsCustomEditControlAction as link.

The TWPToolsCustomEditControlAction is attached to the TWPRichText by a regular ActionList which is referenced by the TWPRichText's property ActionList. Inside the action list you need a TWPToolsCustomEditControlAction for each of the combos. The property AttachedControl must reference the combo, the property AttachedControlStyle is ignored.



TWPToolButton

This components can be used together with a TWPToolPanel. It can control the fonts styles or execute certain actions like 'Print', 'Load' or 'Save'. If you use the TWPToolBar you don't need the TWPCoMboBox or TWPToolButton control since the buttons and combo boxes are created by the TWPToolBar object.



TWPImageList (new)

In the collection Images it is possible to load PNG imagelist as supply for buttons at different button heights. The TWPToolbar will load the best matching images from that collection. The PNG Image list is expected to consists of x buttons arranges one under each other. We created lists of buttons at 24 and 48 pixel height, which will be provided with registered version.

b) Work with Actions



TActionList

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 Version 7. 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. (See ribbon demo)

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))
    and (GetAsyncKeyState(VK_CONTROL) < 0)
  then // Ctrl + I but NOT TAB
  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]).

6.4 Change current writing mode

See property WPRichText.CurrAttr, here you can set attributes such as Bold, Italic etc. using AddStyle and DeleteStyle. [CurrAttr](#) also controls the paragraph attributes (Alignment) indents and spacing. CurrAttr will change the current writing mode or the selected text - if text is selected. See this [FAQ](#) if you want to implement ShortCuts, such a Ctr+B = bold. WPTools Version 7 also has more specialized attribute interfaces. They can be used to only change the current attribute, only the selected text, the attributes of the text which was found by "Finder" or the attributes of merged text. ([Attribute Interface Category](#))

6.5 Change Page Size and Page margins

Modify WPRichText.[Header](#).

A page has 4 margins,
Left, Right, Top, Bottom

Together with PageWidth, PageHeight that defines the page size and printable area.

MarginHeader and MarginFooter define the minimum distance of the page footer text or header text to the borders.

Those properties are all in the sub property Header - see link to manual above - and modify the current document.

The property **DefaultPageWidth** etc. set the values which are applied when the editor is cleared (create new document).

Method SetPageWH can be used to set several values at once, the values which should not be changed can be passed as -1

6.6 Create an own toolbar

If you decide to not use the toolbar we provide please open unit WPCtrRich.PAS and check out method OnToolBarIconSelection.

This method is executed for all toolbar button clicks and WPTools actions. You can see how things are done. You can also call this method from "outside".

When you add standard wptools dialogs to your project, i.e. TWPSearchReplaceDlg You only need to link those to a component of the class TWPDIALOGCollection.



Add a reference to the TWPDIALOGCollection instance to the TWPRichText in property WPDIALOGCollection and the editor can automatically use the dialogs.

6.7 FastAppendText: 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.

6.7.1 Example: 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
          AllRTFText.FastAppendText(WPRichText1,true, [wpCreateNewPage]);
        end;
      Application.ProcessMessages;
      Table1.Next;
      inc(i);
    end;
  finally
    AllRTFText.EndUpdate;
```

```
AllRTFText.ReformatAll(true, true);
Table1.EnableControls;
end;
```

6.7.2 Create Sections with FastAppendText

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
  Section := AllRTFText.FastAppendText(WPRichText1,true,
[wpCreateNewPage]);
  Section.Select := [wpsec_ResetOutlineNums, wpsec_ResetPageNumber];
end;
...
end;
finally
...
end;
```

6.7.3 Create Lists

You can use the flag **wpCombineTableIfPossible** to create a large table from a template which consists only of a single table with fields. If this flag is not used, for each run one table will be created in the destination editor.

If the flag **wpReuseLastPar** was used, empty lines in the destination will be avoided.

Both flags were added to WPTools 7.

6.7.4 Save created text

To save the text use `Dest.SaveToFile` or `Dest.SaveAs`;

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
```

6.8 Get and set the text "as string"

Use the property `AsString` - you can use `SelectionAsString` to read only the selected text. You can also assign text - the property `SelectionAsString` will insert the text at the cursor position.

The **function** `AsANSIString` will work like property `AsString` but you can specify the format string, optionally only save the selection. Also see the [load & save category](#).

6.9 Header and Footer

WTools 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 WTools Version 7 it 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(paraRight));
    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

WTools 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 : TWPMangeHeaderFooter;
```

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

```
if ManHeadFoot=nil then
  ManHeadFoot := TWPMangeHeaderFooter.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 *TWPRichText.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, '');
end
```

```

    // 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.

Copy all header and footer from one editor to another:

```

var i : Integer;
begin
    for i:=0 to WPRichText1.HeaderFooter.Count-1 do
        if WPRichText1.HeaderFooter[i].Kind in [wpIsHeader, wpIsFooter] then
            WPRichText2.HeaderFooter.Get(WPRichText1.HeaderFooter[i].Kind,
                WPRichText1.HeaderFooter[i].Range, '').RtfText.AsString :=
                WPRichText1.HeaderFooter[i].RtfText.AsString;
        end;
    end;
end;

```

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;
```

6.10 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 Version 7 start with a hyperlink start tag `<a>` and end with a closing tag `</a>`. **All these tags are represented by TWPTTextObj instances.** (These tags can be made visible using the 'FormatOptions'!) To read the hyperlinks URL use obj.Source, to modify the visible text change obj.EmbeddedText!

This code can be used to create a hyperlink:

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

Since hyperlinks are marked with paired TWPTTextObj 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 : TWPTTextObj;
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 double 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 files, bookmarks or span codes**. The event OnClickHotText is always triggered by a single click.

Please note that in WPTools Version 7 hyperlinks are not represented by text with a certain attribute (asfHyperlink) followed by hidden text (asfHidden) 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 : TWPTTextObj;
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 sideeffect 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.

6.11 Input TextObjects (i.e. page numbers)

This objects can be create with [InputTextField](#)(type) and InputTextFieldName(name). All text objects can show custom text which is define by event [OnTextObjectGetTextEx](#). Custom text objects will usually display the text assigned to their property Params.

6.12 Insert Text

Use the method [InputString\(\)](#) to insert text at cursor position. The char codes #13 will create a new paragraph, #12 will create a new page.

To create a table use [TableAdd\(\)](#). Also see the chapter [Set Attributes In Code](#).

To load RTF from a string use SelectionAsString := some_rtf_code.

Alternatively LoadFromStream can be used.

6.13 Load & Save

There are multiple methods to load and save text. The high level functions are available in the editor (TWPCustomRTFEdit).

LoadFromFile, LoadFromStream, LoadFromBuffer
SaveToFile, SaveToStream, SaveSelectionToStream

The methods internally use the low level functions defined in the TWPRTFDataCollection class. It is also possible to load the text of a paragraph (TParagraph.LoadFromStream/String) - which is very useful for table paragraphs since it is possible that multiple paragraph can be loaded into one cell.

You can also use the properties AsString and SelectionAsString or the function AsANSIString if you need to work with strings.

The load and save format is modified by "[format strings](#)". This are strings which contain a format name (such as "RTF", "HTML", "AUTO" or the respective reader or writer classname) and optional options, such as "-nobinary" or "-onlybody" which are separated by comma. The function AsANSIString accepts such a format string parameter.

To make the Editor use the DOC import (based on the MS word converter DLLs which may or may not be installed on the system) you need to add the unit WPWordConv to the uses clause. If that unit has been included, the method "Load" will optionally offer the filters which are implemented through the converters.

The property "Text" just extracts plain ANSI text. Use GetUNICODE/SetUNICODE to work with wide strings.

The property Lines.Text should not be used.

The regular save and load methods (LoadFromFile, SaveToFile) can **access text wrapped by paired objects**.

To do so specify the fieldname in the format string, i.e. "f:name=RTF" to save or load the contents of the field "name".

This FormatName uses this syntax to select fields:

"X:text" or "X:text=".

X can be one of this letters:

lowercase letters:

c: search for wpobjCustom comparing the object name = text

f: search for wpobjMergeField

h: search for wpobjHyperlink

b: search for wpobjBookmark

p: search for wpobjTextProtection

s: search for wpobjSPANStyle

capital letters:

C: search for wpobjCustom comparing the object source parameter = text

F: search for wpobjMergeField etc.

If the FormatName was processed the function will return the count of characters which were interpreted.

Also see method `TWPRTFDataCollection.CodeLocatePair`.

6.14 Mail Merge (replace fields with data)

Introduction:

"Mail-merge" means the automatic update of data fields in any document. It can be used to do mass mailing or to create customizable database record views.

This is done by once procedure, [MergeText](#) and the event [OnMailMergeGetText](#). If you have a text with fields marked with special characters (<name>) use method `ReplaceTokens`. Otherwise create fields with method [InputMergeField](#). Note, that mail merge fields are embedded into two instances of the `TWPTextObj` class. This [FAQ](#) shows how to work with images. Note: If you need to print (or export to PDF) the text right after the merge process you need to call `ReformatAll(false, true)`!

Concept of mail-merge in WPTools:

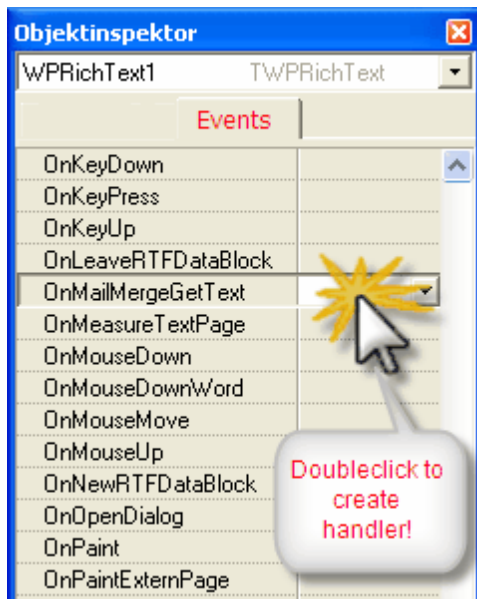
The text contains merge [fields](#), on command "[MergeText](#)" the component locates all fields and triggers the event [OnMailMergeGetText](#) for each of it. This event is used to fill data into the field or read out the current contents. Please note, that this concept differs from the usual "search and replace" and is much more versatile and faster. With WPTools merge fields are not destroyed by the merge process, the field data can be exchanged as soon as it changes. So it is possible to scroll through a database with a merge letter being "attached". It is also possible to read out the contents of the merge field. So the document can be also used as data entry form. You can merge in standard ANSI text, formatted text and images. Formatted text may be encoded in HTML, RTF or the WPTOOLS format. Info: Mail merge templates can be converted to reporting templates which use the WPTools [WPReporter](#) [addon](#).

Three steps to create a working prototype of mail-merge in Your application:

- 1) Add code to create a field ([more...](#))

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    WPRichText1.InputMergeField('NAME', 'Default-Name');  
end;
```

- 2) Provide an **event handler** for the event `OnMailMergeGetText` ([more...](#))



In the created handler please type some lines of code

```

procedure TForm1.WPRichText1MailMergeGetText(Sender: TObject;
  const inspname: String; Contents: TWPMInsertTextContents);
begin
  if inspname='NAME' then
    Contents.StringValue := 'Julian Ziersch'
  else Contents.StringValue := '<unknown>';
end;

```

3) Launch mail merge with a different button ([more...](#))

```

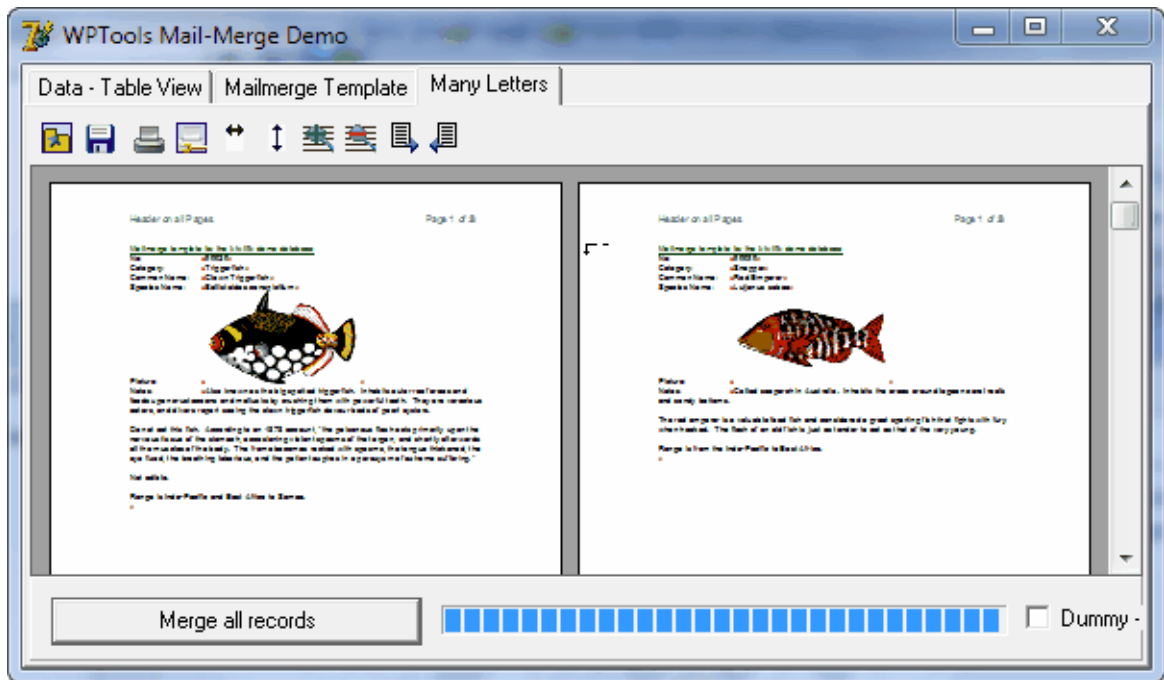
procedure TForm1.Button2Click(Sender: TObject);
begin
  WPRichText1.MergeText;
end;

```

Demo Projects

Please check out the [ThreadSave](#) demo to learn how to use mail merge in a thread.

There is also the demo project **DemosII) MailMergeMailM4** which shows how to work with a database and merge multiple records:



6.14.1 Create Field

[InputMergeField](#) - create a merge field

[InputEditField](#) - create a form field

[ReplaceTokens](#) - convert tokens into fields

[Low Level](#) - create a field in a TParagraph object

6.14.1.1 InputMergeField

To create a field use the procedure **InputMergeField**

```
function InputMergeField(
    const FieldName: string;
    const DisplayText: string = '';
    Command: string = '';
    Format: Integer = 0;
    DisplayChar: Char = #0 //<-- obsolete parameter!
): TWPTTextObj;
```

The parameters are:

FieldName = the name of the field. Will be stored in TWPTTextObj.Name

DisplayText = the text which will be placed inside the field. Will be stored in TWPTTextObj.

Params. Can be modified in the editor.

Command = any text which will be stored in TWPTTextObj.Source

Format = any integer which will be stored in TWPTTextObj.IParam

DisplayChar should be not used anymore.

The result of this function is a reference to the **TWPTTextObj** which marks the start of the field.

Please note, a merge field always consists of two object, a start and an end object. Only the start object contains the properties of the field.

WPTTextObj instances can be deleted and restored at any time by the editor. It is not recommended to save the references in a list. To fill a list with reference to field objects you can use the procedure FieldGetList at any time. You can also use the 'Code' API with field objects - for example the

procedure CodeListTags. (See reference, HLP)

6.14.1.2 InputEditField

WPTools Version 7 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 fields use the Method [InputEditField](#).

[More about forms ...](#)

6.14.1.3 ReplaceTokens

ReplaceTokens(const opening, closing: string; FieldPreText: string = ""): Integer;

This method (and its sister ReplaceTokensInAllTexts) updates the text and converts tokens such as **<NAME>** into the respective field. The start and end character sequence is variable, you have to pass it to the function: `ReplaceTokens('<', '>');`

The third parameter is the text which will be used to create the field name. You can use it to create fields such as "CUSTOMER.NAME" by using 'CUSTOMER.' as parameter FieldPreText.

ReplaceTokensInAllTexts does not only work in the current text but in all, header and footer texts.

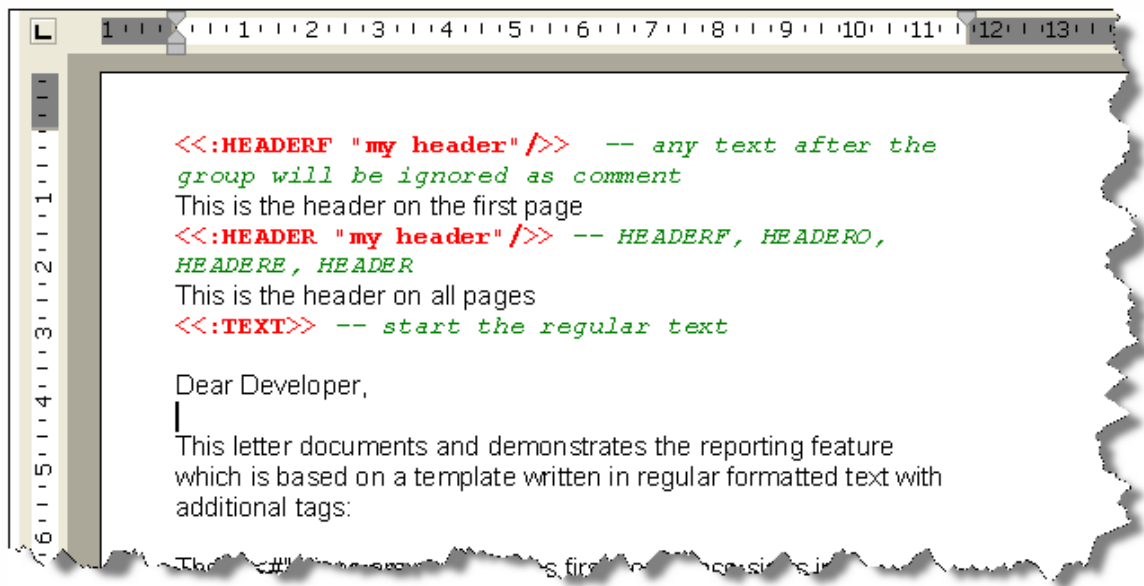
Both function return the count of replacements.

Tip: **WPTools 7 with the WPReporter addon also supports "Token to Template Conversion"**.

If you use the token to template conversion you can edit the mail-merge/reporting template with an editor such as MS Word.

The template is loaded into TWPRichText. In this control the template can be further edited with the additional convenience of syntax highlighting and on demand converted into a "true" reporting template. In case no bands are used, this technique can be also used to create mail merge templates. (WPTools' Reporting is backward compatible to mail merge)

Example:



6.14.1.4 Low Level

If you work with TParagraph references You maybe want to create fields using low level routines.

There are two possibilities.

a) Use TParagraph.AppendNewObject

```

var starto, endo : TWPTextObj;
    par : TParagraph;
begin
    par := WPRichText1.ActiveParagraph;

    starto := par.AppendNewObject(wpobjMergeField, true, false); // Start Object
    par.Append('Julian Ziersch'); // Displayed Text
    endo := par.AppendNewObject(wpobjMergeField, true, true); // End Object
    // Link both objects together
    endo.SetTag(starto.NewTag);
    // Set Name of field
    starto.Name := 'DB_NAME';

    // Reformat and display
    WPRichText1.ReformatAll(false, true);
end;

```

b) Use TParagraph.AppendNewObjectPair

```

var fieldo : TWPTextObj;
    par : TParagraph;
begin
    par := WPRichText1.ActiveParagraph;

    fieldo := par.AppendNewObjectPair(wpobjMergeField, 'Julian Ziersch');
    fieldo.Name := 'DB_NAME';
    // Reformat and display
    WPRichText1.ReformatAll(false, true);
end;

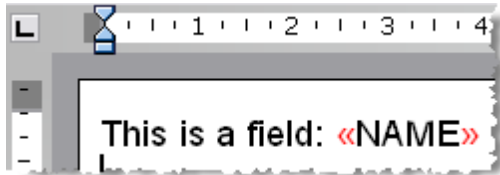
```

Using method (a) You can initialize the field with formatted text by subsequently adding text and changing the current writing mode between the text parts.

6.14.2 Customize Field Display

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`.

By default this objects are displayed like in this image:



The display of the markers is optional. Please set the property `WPRichText.InsertPointTextAttr.Hidden = true` to hide the field markers. `WPRichText.InsertPointTextAttr.Hidden` should be set to true when the document is printed. To permanently delete the fields (and keep the text) use the procedure `DeleteFields`.

This method will toggle display depending on the state of a check box:

```
procedure TForm1.ShowFieldsClick(Sender: TObject);
begin
    if ShowFields.Checked then
    begin
        wprichtext1.InsertPointAttr.hidden:=false;
        wprichtext1.automatictextattr.BackgroundColor := clYellow;
        wprichtext1.automatictextattr.UseBackgroundColor := TRUE;
    end else
    begin
        wprichtext1.InsertPointAttr.hidden:=true;
        wprichtext1.automatictextattr.UseBackgroundColor := FALSE;
    end;
    wprichtext1.ReformatAll(false, true);
end;
```

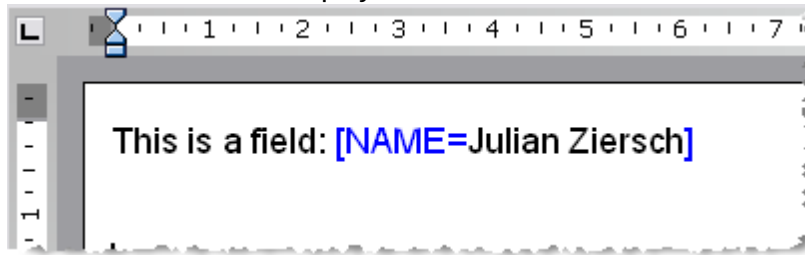
It is also possible to show a different text in a different color.

The text is defined by the public (not published) properties `CodeOpeningText` and `CodeClosingText`. The variables `%N`, `%S`, `%Y` and `%P` can be used. The color can be changed with property `CodeTextColor`.

```
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
```

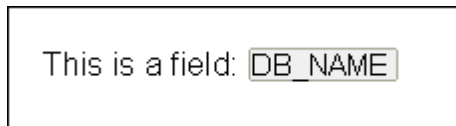
```
WPRichText1.InsertPointAttr.CodeClosingText := ']';  
WPRichText1.InsertPointAttr.CodeTextColor := clBlue;
```

This is how the field is displayed now:



Display Fieldnames only:

WPTools 7 also allows to hide the embedded text and the fieldmarker and display just a small box with the field marker:



This mode is controlled by property **ShowMergeFieldNames**. The value TRUE will show the boxes while the value FALSE will enabled the default display (with the field markers visible or not).

6.14.3 Update Field (Insert Text from Database)

The mail merge process activates some or all all fields one after the other and calls an event to let the code fill in text and images in each of it.

6.14.3.1 Start Merge process

The mail merge is **started** by the method

```
MergeText(const FieldName: string = "; AllTexts: Boolean = FALSE).
```

Both parameters are optional. If you need to merge only the text the cursor is located in (header, footer ..) use procedure MergeActiveText;

Example:

```
MergeText('', true) - merge the fields in all areas, headers and footers
```

```
MergeText('A*', true) - merge the fields with names starting with "A" in all areas, headers and footers
```

```
MergeText('A*', 'name', true) - merge the fields with names starting with "A" and Source='name' in all areas, headers and footers
```

Important:

If you need to print (or export to PDF) the text right after the merge process you need to call `ReformatAll(false, true)`!

If you need to merge the fields **only in a header or footer** text use the MailMerge procedure of any `TWPRTFDataBlock`. You will have to pass the `OnMailMergeGetText` procedure described below as callback.

Example:

Merge fields only in all header texts:

```
for i:=0 to WPRichText1.HeaderFooter.Count-1 do
if (WPRichText1.HeaderFooter[i].Kind=wpIsHeader) and // only header
not (WPRichText1.HeaderFooter[i].IsEmpty) then // and not empty
WPRichText1.HeaderFooter[i].MergeText(
WPRichText1, // object used as 'Sender', can be also
datasource ...
WPRichText1MailMergeGetText, // event handler (see below)
false, // readonly, we want to modify
'', nil, nil ); // optional to restrict merge range
```

6.14.3.2 Event OnMailMergeGetText

This event is used to modify a field. It can also be used to enumerate fields and also to reads the properties of the field or the embedded text.

The event `OnMailMergeGetText` receives the following parameters:

Sender: TObject - this is a reference to the parent editor

const inspname: String - the name of the field

Contents: TWPMInsertTextContents - this is the most important parameter. It provides You with an interface to evaluate and manipulate the field and the embedded text.

In the easiest case You can use one line of code to assign text to a field:

```
procedure TForm1.DoMailMergGetTexte(
Sender: TObject; const inspname: string;
Contents: TWPMInsertTextContents);
begin
Contents.StringValue := DataSet.FieldByName(inspname).AsString;
end;
```

This code will read the contents of a given field and use it as the embedded text of a field. If the field contains HTML or RTF formatted text the text will be inserted as formatted text.

If you need to **modify the character attributes** (such as font name, font style and color) of the field inside this event, please use the interface [MergeAttr](#).

**Attention: Do NOT call ReformatAll,
do not move the cursor position,
do not change InsertPointTextAttr.Hidden inside this event!**

Example: Display a checkbox:

```
Contents.MergeAttr.SetFontName
if data=true then
Contents.StringValue := #254 // ☑
else Contents.StringValue := #168; // ☐
```

6.14.3.2.1 TWPMInsertTextContents

The TWPMInsertTextContents is used as a parameter 'Contents' in the event OnMailMergeGetText. You can replace a string by simply assign a string to StringValue.

To Modify the attribute of the inserted text (i.e. the font name or style) change the property 'MergeAttr'.

You can modify the paragraph the start of the field is located. This paragraph is accessible through MergePar.

Upgrade Note: The parameter 'c' is not supported anymore.

But please note that you can use the common programming interface to insert text and objects.

Note: When the event is called the the whole field is selected.

6.14.3.2.1.1 procedure Clear

Clears the contents of this object.

6.14.3.2.1.2 procedure Abort

Notifies the merge routine to abort the process.

6.14.3.2.1.3 procedure LoadTextFromStream(s: TStream)

Load text from a stream.

6.14.3.2.1.4 procedure LoadTextFromFile(const FileName: string)

Load text from a file.

6.14.3.2.1.5 function LoadImageFromFile(const FileName: string; w: Integer = 0; h: Integer = 0): Boolean;

Load an image into this field. This methods have been added to WPTools 7.

If You specify the width and height (twips values) the image will be scaled, keeping the original aspect ratio.

You can pass negative values for w and h to preserve the width and height of the image currently embedded in the field.

If currently no image is in the field, w and h will be used as absolute values.

```
if inspname='IMAGE_A' then
begin
  Contents.LoadImageFromFile('c:\a.jpg', -1440, -1440);
end;
```

Also available is

function LoadImageFromStream(const ImageData: TStream; FileExt: string; w: Integer = 0; h: Integer = 0): Boolean;

It expects the data in a stream object and the file extension which defines the contents of the stream. For example ".JPG" if it is JPEG data.

6.14.3.2.1.6 property DataCollection: TWPRTFDataCollection

This is the current RTFDataCollection

6.14.3.2.1.7 property DataBlock: TWPRTFDataBlock

The is the DataBlock which hosts the current paragraph

6.14.3.2.1.8 property MergeAttr: TWStoredCharAttrInterface

This interface allows it to modify the attributes of the text which is about to be inserted.

Also see: "[Preserve Attributes of text inside a field](#)"

You can use the methods of the [AttributeInterfaces](#).

Example:

```
Contents.MergeAttr.SetColor(c1Red);
```

The object 'MergeAttr' is initialized with the character properties of the start object of the mailmerge field. If you want to use the first character of the merged text (this is the visible field name or the field data) You can use this code in the the OnMailMergeGetText event:

```
Contents.MergeAttr.CharAttr :=
  Contents.MergePar.CharAttr[
    Contents.MergeParPos+1];
```

This code assigns the character attribute of the character which follows the the mergefield start code (= the merge text).

You can, for example, change the font name:

```
Contents.MergeAttr.SetFontName(...);
```

This can be very useful if you need to **display checkboxes in the fields**. In this case simply assign the desired wingdings characters code to StringValue.

```
Contents.MergeAttr.SetFontName
if data=true then
  Contents.StringValue := #254 // ☒
else Contents.StringValue := #168; // ☐
```

6.14.3.2.1.9 property MergePar: TParagraph

MergePar is the paragraph the merged field starts within. [MergeParPos](#) is its position.

6.14.3.2.1.10 property MergeParPos: Integer

[MergePar](#) is the paragraph the merged field starts within. MergeParPos is its position.

6.14.3.2.1.11 property Obj: TObject

If an object of type TWPObj is assigned to this value, the new object is inserted into the text. This can be used to merge images.
More easy to use is LoadImageFromFile and LoadImageFromStream.

6.14.3.2.1.12 property CodePage: Integer

This CodePage, if $\neq 0$, the value will be used to convert the string value to the internal representation.

6.14.3.2.1.13 property StringValue: AnsiString

This is the result string as ANSI string. You can use the property CodePage to specify which code page should be used to convert it to the intern UNICODE representation. Please also see WideStringValue.

You can also assign formatted text in RTF, HTML or WPTOOLS format to StringValue.

If you merge in RTF text and the inserted text starts with a table you will see an empty line. This "empty" line is used to store the field marker which is not automatically deleted by the merge process. If it is not required to repeat the merge process or to read out modified text you can let the Merge procedure delete the field marker. Simply set the flag **mmDeleteThisField** in the property *Contents.Options*. This flag can be also very useful if the merged text contains field markers on its own which have to be processed in a second run!

6.14.3.2.1.14 property WideStringValue: WideString

This is the result string as unicode string. If you need to insert formatted text (RTF, HTML) please use StringValue.

6.14.3.2.1.15 property OldText: string

This is the current text inside the field.

6.14.3.2.1.16 property OldFormattedText: string

This is the current text inside the field including formatting tags.

6.14.3.2.1.17 property OldIsPlain: Boolean

This value is true if the "[OldText](#)" does not use different character attributes.

6.14.3.2.1.18 property CurrentObject: TWPObj

This is a reference to the object currently INSIDE the field. It can be used to update a graphic container.

6.14.3.2.1.19 property CurrentTxtObject: TWPTxtObj

This is a reference to the object currently INSIDE the field. It can be used to update a graphic container.

6.14.3.2.1.20 property Options: TWPMailMergeContinueOptions

This SET contains flags which change the way the data is inserted and how the method proceeds.

mmStopNow, aborts the merge process
 mmMergeTrim, // WPTools 7: removes leading and trailing spaces from the replacement string.
 (Not used for formatted text)
 mmUseFirstLoadedParProps, // When loading RTF or HTML we also assign the paragraph props
 of the first loaded paragraph (WPT6: Use /setattr in Extras)
 mmUseFirstLoadedParPropsAuto, // use mmUseFirstLoadedParProps if the field is the first NON
 SPACE sign in the paragraph
 mmUseFirst_AlsoBorders, // refines "mmUseFirstLoadedParProps" to copy also the border and
 padding properties
 mmIgnoreNewParAttr, // If we are loaded in RTF or HTML string then paragraph attributes are
 preserved (WPT6: Use /keepattr in Extras)
 mmIgnoreNewCharAttr, // If we are loaded in RTF or HTML string then character attributes are
 preserved
 mmIgnoreLoadedFonts, // When loading RTF ignore the Fonts
 mmIgnoreLoadedFontSize, // When loading RTF ignore the Font sizes
 mmIgnoreLoadedFontStyles, // When loading RTF ignore bold/italic and underline
 mmMergeAsRTF,
 mmMergeAsHTML,
 mmMergeAsWPTOOLS,
 mmSkipSpacesBehind, // not used yet
 mmInsertObject,
 mmDeleteEmptyParagraph, // not used yet
 mmDeleteUntilFieldEnd, // not used yet
 mmDeleteThisField, //Delete the field markers! (WPT6: Use /remove in Extras)
 mmDontUseLoadedParTabs, // When loading RTF optionally IGNORE the tabs
 mmDontUseLoadedParStyle // When loading ignore the style

, mmMergeTrimSpaces //WPTools 7: remove double spaces between words, can be combined
 with mmMergeTrim (Not used for formatted text)

6.14.3.2.1.21 property Name: string

This is the field name.

6.14.3.2.1.22 property FieldnamePart: string

This is the part of the string NAME after the character '.', converted to uppercase characters. If there
 is no '.', 'Name' will be returned.

6.14.3.2.1.23 property DatasetnamePart: string

This is the part of the string 'NAME' before the character '.', converted to uppercase characters. If
 there is no '.' DatasetnamePart will be empty.

6.14.3.2.1.24 property Command: string

This is the command - it is placed in the "Source" property of the field object.

6.14.3.2.1.25 property StartInspObject: TWPTTextObj

The reference to the starting merge field object.

6.14.3.2.1.26 property EndInspObject: TWPTTextObj

The reference to the closing merge field object.

6.14.3.2.1.27 property DisplayName: string

Reserved

6.14.3.2.1.28 property Format: Integer

Reserved

6.14.3.2.1.29 property Modified: Boolean

This value is true if a value has been assigned to StringValue or Obj.

6.14.3.2.1.30 property Hyperlink: string

If this property is not empty, a hyperlink with this URL and the name "[HyperlinkName](#)" will be created.

6.14.3.2.1.31 property HyperlinkName: string

The name of the hyperlink with the URL "[Hyperlink](#)".

6.14.3.2.1.32 property IsNull: TWPMInsertTextContentsNull

This property is only used for reporting in WPTools 7.

This new property can be used to provide the WPREporter engine with additional information about a field. This is especially useful when a field is used for a condition. The standard value is wpFieldDefault - here the contents will be evaluated using the StringValue and Modified properties.

```
wpFieldDefault, // Preset when reading field data
wpFieldIsUnknown, // Preset when conditions are checked (?...=...)
wpFieldIsNull, // If this is set any value will be ignored!
wpFieldIsZero, // Should be set if an integer or float value = 0
wpFieldIsDefined // Should be set if field is defined
```

6.14.3.2.1.33 property ReplaceMode: TWPMInsertTextContentsReplaceMode

```
wpconNoChange,
wpconStringValue,
wpconWideStringValue,
wpconObject,
wpconStream
```

6.14.3.2.2 Preserve Attributes of text inside a field

Normally, while merging in ANSI text (in contrast to RTF text), the inserted text will get the attributes of the first field marker.

If the text inside the field markers is specially formatted, this format properties are usually lost.

But it is possible to force the inserted text to use certain attributes. This is done with property

[MergeAttr.](#)

So, just a few lines can be used to read the character attributes of the first character in the field:

```
Contents.MergeAttr.CharAttr :=
  Contents.StartInspObject.ParentPar.CharAttr
  [ Contents.StartInspObject.ParentPosInPar + 1 ];
```

Here we read the CharAttr index of the character after the field marker "StartInspObject".

This code has the problem that it does not check, if the field marker happens to be the last character in a paragraph. This is not usual, but it can happen.

So better take advantage from the fact, that the cursor is before the first character in the field during OnMailMergeGetText.

This code will work to read the first character attributes:

```
Contents.MergeAttr.CharAttr := WPRichText1.CPCharAttr;
```

6.14.3.3 FieldLocate: Modify Field Name or Contents

The powerful function named **FieldLocate** can be used as alternative to the MergeText/OnMailMergeGetText double. It is used to locate certain fields.

This code will enumerate a certain group of fields and update their contents.

```
obj := nil;
FromStart := TRUE;
repeat
  obj := WPRichText1.FieldLocate('A*B', '',
    FromStart,
    [wplocGlobalSearch,wplocDontMoveCursor]);
  if obj<>nil then
    obj.Embeddedtext := 'Test';
    FromStart := FALSE;
until obj = nil;
```

An alternative method to enumerate the fields is the function **FieldGetList**. It will fill a list (an instance of the TWPTTextObjList class) with references of fields. This list can be used to change the names of the fields and also to read or change the contents (EmbeddedText).

The property **FieldAtCP** can be used to retrieve a reference to the active field at the current cursor position. Please use this function with care since the backward search for the open object can be slow, especially if no field can be found. CPObj is faster, it returns any TWPTTextObj directly at the current cursor position (= at the current text insertion point - do not confuse with mouse-cursor).

6.14.3.4 Use TWPMMDDataProvider

This component can be used to automatically attach a mail merge template to a data set. The property `NextDataProvider` can be used to build a chain of data provider.

The event `OnGetDataSet` can be used to locate the correct dataset for a given field. (i.e. evaluate [Contents.DataSetNamePart](#))

Fieldsnames which are listed in the list `BMPFields` are expected to be pictures, the fields in `RTFFields` are expected to contain formatting tags.

If a field is of type `TGraphicField`, automatically the contained image will be inserted. Here the flag `wpmmPreserveObjectSize` can be used in property "Options" to load the image into the current container (if there is one).

Info: The data merging is performed by

```
procedure TWPMMDDataProvider.DoMergeGetText(Sender: TObject;
  const fieldname: string;
  Contents: TWPMMDInsertTextContents);
```

found in unit `wpdbrich.pas`. We recommend to read it.

6.14.4 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

```

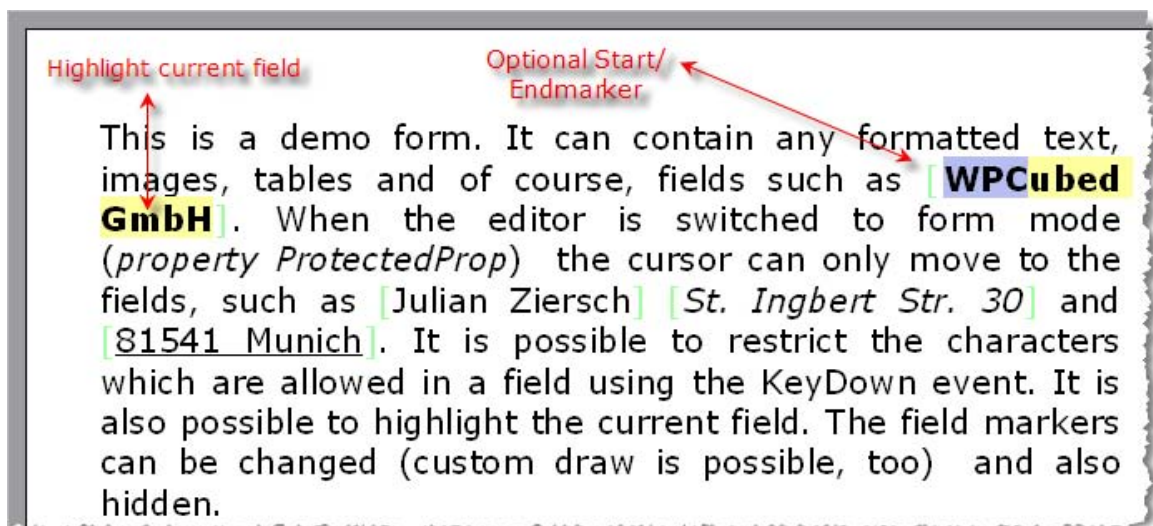
function DeleteLeadingSpace(EmptyFieldsToo: Boolean;
InFirstPar : Boolean = TRUE): Boolean;
function DeleteTrailingSpace(EmptyFieldsToo: Boolean):
Boolean;

```

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.

6.14.5 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 TWPTTextObj.

```

Source

```

  Format: Integer = 0 // will be written to TWPTTextObj.

```

IParam

```

): TWPTTextObj;

```

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:

```

DataEditAutomaticTextAttr.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: TWPMMinertTextContents);
begin
    if FReadingData then
        Table1.FieldName(inspname).AsString := Contents.
OldText

```

```

        else Contents.StringValue := Table1.FieldName(inspname).
AsString;
    end;
end;

```

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

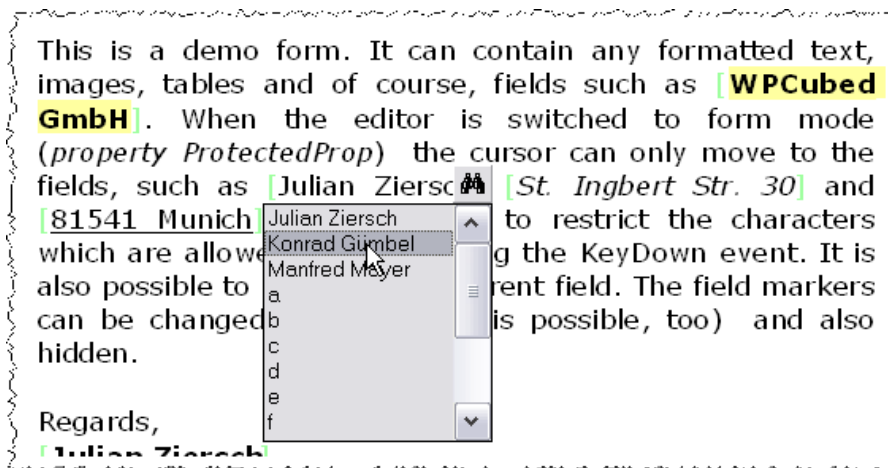
```

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;

```

6.14.6 Create ComboBox

Now we show how to create a pick list for a merge field:



We use a panel with a speed button to display the drop down button. This panel is moved to the correct location in the MouseMove event. A timer is used to hide this panel if it is not required anymore. The picklist itself is a listbox, also placed on the form. It would be better to display this list on a separate form located over the mainform but we wanted to make this example as simple as possible.

The name of the current field (for which the picklist is displayed) is stored in the string variable CurrField.

The **MouseMove** event handler for the editor. It sets 'CurrField'

```

procedure TWPEdTest.DataEditMouseMove(Sender: TObject; Shift:
TShiftState;
  X, Y: Integer);
var fieldobj, fieldobjend : TWPTextObj;   px, py : Integer;
begin
  fieldobjend := nil;
  fieldobj := DataEdit.CodeInsideOf(x, y, wpobjMergeField);
  if (fieldobj <> nil) and ((CompareText(fieldobj.Name, 'name')=0
) or
                                (CompareText(fieldobj.Name, 'company'
)=0)) then
    begin
      fieldobjend := fieldobj.EndTag;
      if fieldobjend<>nil then
        begin
          DataEdit.GetParXYBaselineScreen(fieldobjend.ParentPar,
fieldobjend.ParentPosInPar, px, py ) ;
          PickPanel.Left := px - PickPanel.Width div 2;
          PickPanel.Top  := DataEdit.Top + py - MulDiv(PickPanel.
Height,4,5);
          CurrField := fieldobj.Name;
        end;
      end;
      if fieldobjend=nil then
        HidePickPanelTimer.Enabled := TRUE // we hide it 500ms
delayed
      else
        begin
          PickPanel.Visible := true;
          HidePickPanelTimer.Enabled := FALSE;
        end;
      end;
    end;

```

The timer *HidePickPanelTimer* disables the button

```

procedure TWPEdTest.HidePickPanelTimerTimer(Sender: TObject);
begin
  HidePickPanelTimer.Enabled := FALSE;
  PickPanel.Visible := false;
end;

```

The **OnExit** event of the listbox (it is called when the listbox loses the focus) is used to hide the listbox when the user clicks on the editor:

```

procedure TWPEdTest.PicklistExit(Sender: TObject);
begin
  Picklist.Visible := FALSE;
end;

```

A click on the speedbutton display the listbox:

```

procedure TWPEdTest.PickDropClick(Sender: TObject);
begin
    Picklist.Left := PickPanel.Left + PickPanel.Width - Picklist.
Width;

    if CompareText(CurrField, 'name')=0 then
        Picklist.Items.Assign(PicName.Lines)
    else if CompareText(CurrField, 'company')=0 then
        Picklist.Items.Assign(PicCompany.Lines)
    else Picklist.Items.Clear;

    if PickPanel.Top>EditTab.Height-100 then
    begin
        Picklist.Height := PickPanel.Top -20;
        if Picklist.Height>150 then Picklist.Height := 150;
        Picklist.Top      := PickPanel.Top - Picklist.Height;
    end else
    begin
        Picklist.Top      := PickPanel.Top + PickPanel.Height;
        Picklist.Height := EditTab.Height - Picklist.Top-20;
        if Picklist.Height>150 then Picklist.Height := 150;
    end;
    Picklist.Visible := TRUE;
    Picklist.SetFocus;
end;

```

Last but not least - a click in the listbox should update the current field. We use the function `[b]CodeListTags[/b]` to find all the field with a given name and simply update the **property EmbeddedText**.

```

procedure TWPEdTest.PicklistClick(Sender: TObject);
var objlst : TWPTextObjList;
    i : Integer;
    s : string;
begin
    if Picklist.ItemIndex>=0 then
        s := Picklist.Items[Picklist.ItemIndex]
    else exit;
    objlst := DataEdit.CodeListTags(wpobjMergeField, CurrField,
true);
    try
        for i:=0 to objlst.Count-1 do
            objlst[i].EmbeddedText := s;
        finally
            objlst.Free;
            DataEdit.ReformatAll(false, true);
        end;
        DataEdit.Setfocus; // also hides Picklist!
    end;

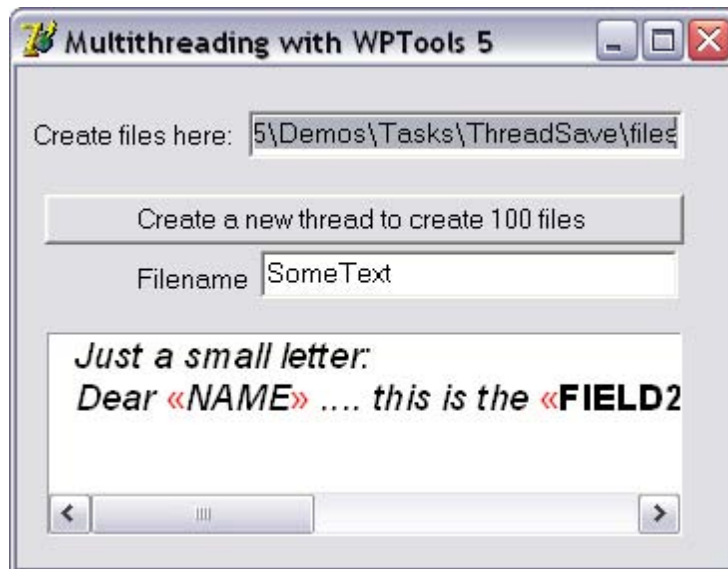
```

You can download the complete example here
<http://www.wpcubed.com/ftp/ex/EditFieldDemo.zip>

6.14.7 Working with multiple threads

Unlike many other editor components WPTools Version 7 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 being 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 separate 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;
  {$IFDEF NOENVIROMENT}
  Enviroment := TWPToolsEnviroment.Create(nil);
  Enviroment.Assign(GlobalWPToolsCustomEnviroment);
  RichText.Memo.RTFData.RTFProps.Enviroment := Enviroment;
  {$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 threadsafe 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**!

6.15 Move the Cursor

Please use the [CP..](#) properties. [CPPosition](#) is the current character position, assign 0 to go to the start, MaxInt to go to the end. Also procedure MovePosition is very useful. When the user changes the cursor position the event OnChangeCursorPos is triggered.

Please do not use CPPosition := CPPoition +1 to move through the text. This is not effective. Please use CPMoveNext instead.

If you need to modify multiple paragraphs in code it is better to create a loop which locates all paragraphs by using the internal pointer tree.

Example - will insert Quotation marks as used in emails: '>>'

```

var par : TParagraph;
begin
  par := WPRichText1.FirstPar;
  while par<>nil do
  begin
    // Do something with par
    par.Insert(0, '>> ',0);
  end;
end;

```

```

    // Next paragraph
    par := par.next;
end;
// Make sure the text is formatted
WPRichText1.ReformatAll(false, true);
end;

```

6.15.1 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. (Also read about the cursor class [TWPRTFDataCursor](#))

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;
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.

Please note when you use WPTools 4 before:

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.

6.15.2 Search&Replace 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.

Overview TWPTTextFinder:

Method Clear - resets the attributes

Method DropMarkerAtFoundPosition

This function drops a cursor marker, see DropMarker. The optional parameter offset will be added to the 'position in paragraph'.

Example - extract text in brackets []:

```
var startid, endid : Integer;

with WPRichText1.Finder do
begin
  ToStart;
  while true do
  begin
    if not Next('[') then break;
    startid := DropMarkerAtFoundPosition(1);
    if not WPRichText1.Finder.Next(']') then break;
    endid := DropMarkerAtFoundPosition(0);
    WPRichText1.TextCursor.SelectMarker(startid, endid);

    ShowMessage(WPRichText1.AsANSIString('ANSI', true));

    WPRichText1.TextCursor.CollectAllMarker;
  end;
end;
```

Method FindAgain - uses the last search text

Method MoveToFoundPositionEnd - moves cursor

Method MoveToFoundPositionStart - moves cursor

Method **Next** - searches a text - returns TRUE if successful.

Method Prev - searches a text backwards - returns TRUE if successful.

Method ReplaceAll - replaces text

Method ReplaceAllW - replaces text using unicode strings

Method SelectText - selects the text which was found

Method SetFoundImage - replaces the found text with an image

Property Position - current position where the search starts.

Method ToEnd - current position - goto end

Method **ToStart** - current position - goto start

Property CaseSensitive - true or false

Property CharAttr - If attributes have been defined in this property the attributes of the text which is found must contain this attributes. Please make sure you clear this property with CharAttr.Clear after your code has been processed!

Property EndAtSpace

This property restricts the wildcard search to stop when the next space or object is found. Also see EndAtWord.

This example creates hyperlinks for all texts which start with http://:

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

Property EndAtWord

This property restricts the wildcard search to stop when the next word delimiter is found.

Property Found

This property is true after the Next found the text. It is not update by ReplaceAll. The method ToStart resets this value to FALSE.

Property FoundLength - length of the found text

Property FoundParagraph - the paragraph wher the text was found

Property FoundPosInPar - the position where the text was found in 'FoundParagraph'

Property FoundPosition - The absolute character positiuon of the found text. Can be used to initialize the property WPRichText.CPPosition. Better use MoveToFoundPositionStart.

Property **FoundText** - Reads and replaces the found text with new text. (Dont' forget WPRichText.DelayedReformat).

Please note that it is not possible to insert new paragraphs using this property. If you need to insert paragraphs or formatted text use SelectText and assign the new text to TWPRichText.SelectionAsString.

Property **FoundAttr**

Reads and changes the attributes of the found text..

Property WholeWord - if true the found text must be within white spaces

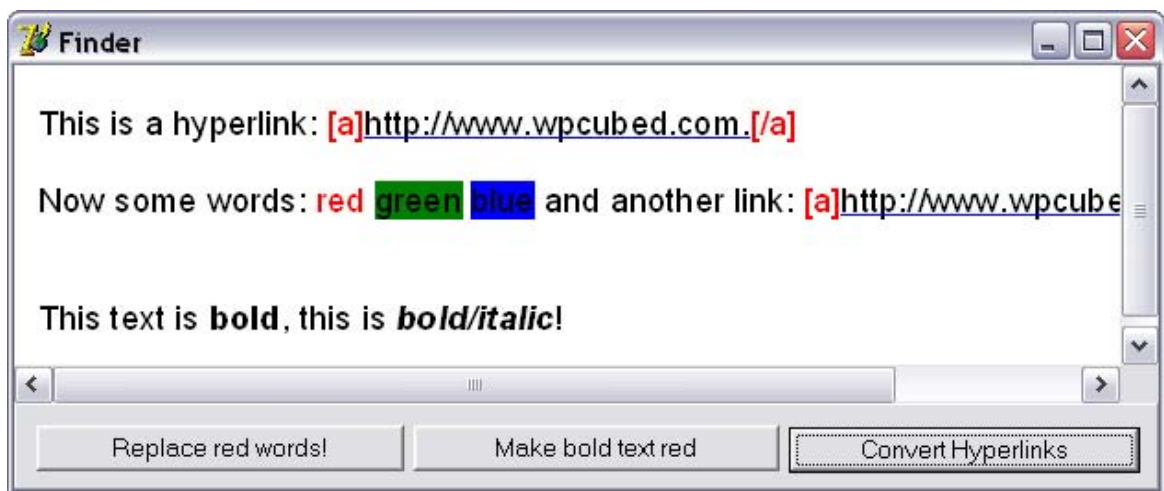
Property WildCard - the wild card character allowed in the search string, for example '*'

Example: Replace the placeholder [NAME] with data using a bold font:

```
WPRichText1.Finder.ToStart;
while WPRichText1.Finder.Next(' [NAME] ') do
begin
    WPRichText1.Finder.FoundAttr.IncludeStyle(afsBold);
    WPRichText1.Finder.FoundText := 'Julian Ziersch';
end;
WPRichText1.DelayedReformat;
```

Example: Convert Hyperlink:

The "Finder" demo project 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)

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

Example: 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 := '*';
```

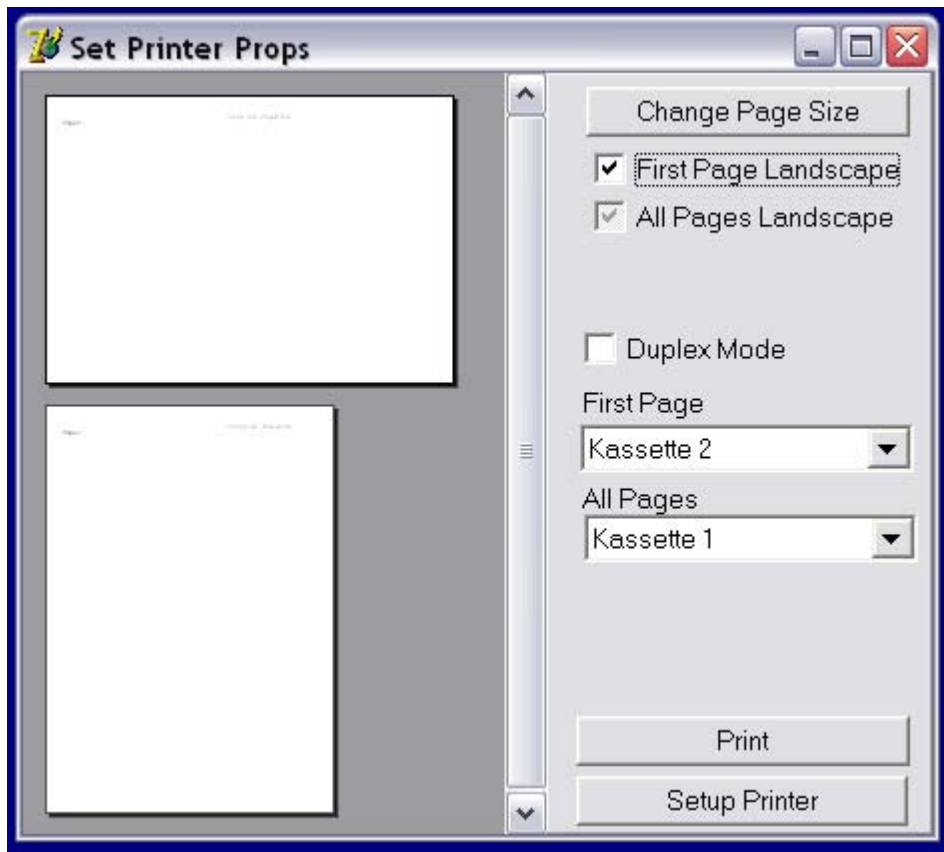
```
while Finder.Next('*') do
begin
    Finder.FoundText := 'Test';
    Finder.FoundAttr.SetColor(clBlack);
end;
Finder.CharAttr.Clear;
WPRichText1.Refresh;
end;
```

6.16 Printing - how to set printer properties

WPTools Version 7 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 '**H) Techniques\PrinterSetup**' 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.Environment.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 Version 7 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 printtime, 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 property `InsertPointTextAttr.Hidden` if this flag is not true anyways.

6.17 Save and load HTML

HTML is autotected when using the LoadFromFile or LoadFromStream methods. Images must be directly linked to local files. Otherwise the image must be loaded in the event [OnRequestHTTPImage](#). (This event is also executed for images in RTF files which have not been embedded but just linked.) Also see this [FAQ](#) and the info about [format strings](#).

6.18 Save to HTML / Load from HTML

WPTools will create a HTML file if you save to a file with extension HTML or HTM. To get the HTML code as a string use `WPRichText1.AsANSIString('HTML');`

'HTML' is a "format string" - please read more about this option strings [here](#).

If your text contains embedded images you will need to save them to a file during the creation of the HTML code.

This FAQ shows how to do it: <http://wpcubed.com/forum/viewtopic.php?t=1167>

This FAQ shows how to load from HTML and include images: <http://wpcubed.com/forum/viewtopic.php?t=1498>

6.19 WYSIWYG

The word WYSIWYG abbreviates **What You See Is What You Get** - which means that the printed output of an application which supports WYSIWYG will match the screen output it displayed before. **WPTools Version 7 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 WYSIWYG *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.UpdateReformatMode( true )`. Now the RTF-Engine will use the current printer to

measure the fonts. For special printer fonts this can make a big difference for the output quality.

To work with mode (c) you can simply set the flag `wpfAlwaysFormatWithScreenRes` in the property `FormatOptions` of the `TWPRichText`.

If you know that there is no printer available for the application, you can set the global boolean **WPNoPrinterInstalled** to true. This property is automatically initialized by using the windows `EnumPrinters()` API.

Please note that the **property WordWrap** disables WYSIWYG. If this property is set to TRUE this means that the text is formatted to the width of the editor, not the paper size defined in the property `Header` or the section. When `WordWrap` is set to false, word wrapping is still performed, but not using the width of the editor but the defined page size. You can, of course, set `WordWrap` to false and still modify the page size dynamically (`Header.PageWidth` according to the size of the editor. In this case WYSIWYG printing is possible, but you probably need to set the flag `PrintParameter.PrintOptions := [wpDoNotChangePrinterDefaults]` - otherwise the printer will select a custom paper size.

Upgrade note: There are no properties `ScreenResMode` and `WYSIWYG` in WPTools Version 7, 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. If the current printer is changed the reformat of the text is not required.



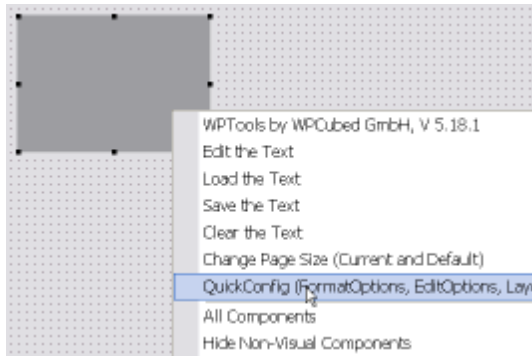
WPTools supports many different layout modes. It is possible to see the page with header and footers, several pages side by side, thumbnails, just the body text etc. Please read the chapter about [LayoutModes](#).

7 Guide

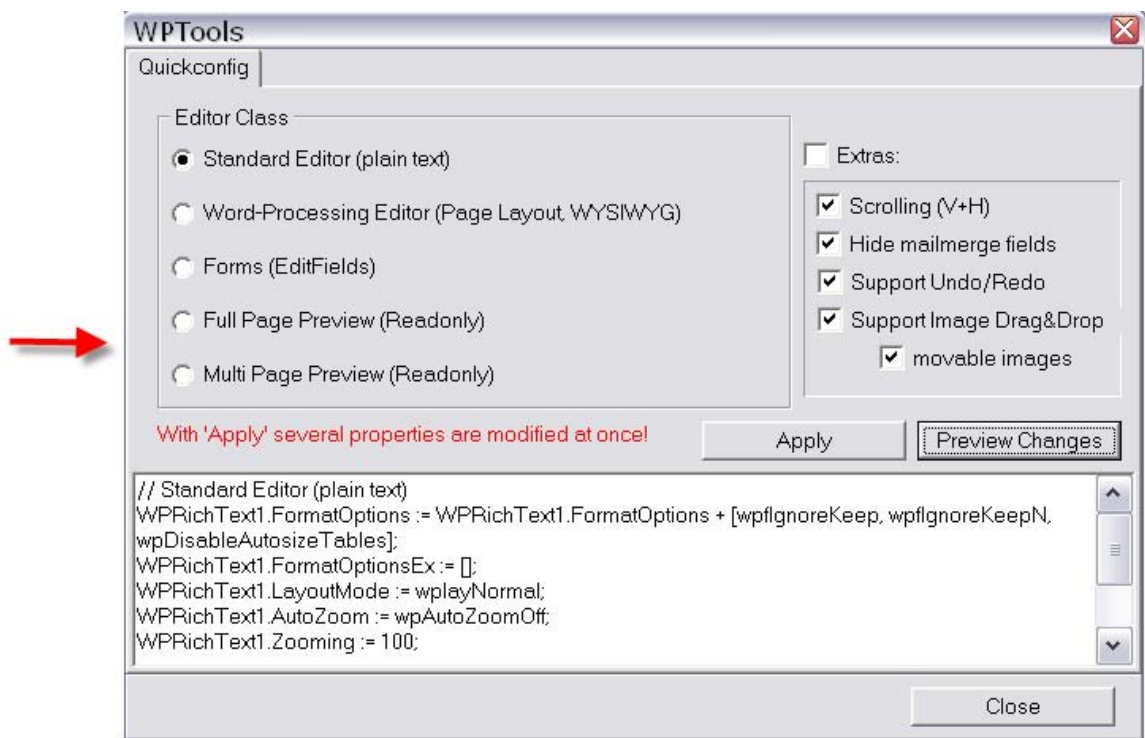
In this chapter we collected introduction texts to important tasks which can be solved with WPTools. We recommend to also review the HLP file since it contains a structured list of all classes, properties and methods. It also contains 'Categories' which are a big help.

QuickStart

After having dropped the TWPRichText you can **click right** to use the QuickConfig dialog:



[the WPTools Property Dialog]



TWPRichText - Property Overview

AcceptFiles	True		
ActionList	ActionList1		
Align	alClient		
AllowMultiView	False		
Anchors	[akLeft,akTop,akRight,akBottom]		
AutomaticTextAttr	(TCharacterAttr)		
AutoZoom	wpAutoZoomOff		
BookmarkTextAttr	(TCharacterAttrTags)		
BorderStyle	bsNone		
ClickableCodes	[wpobjMergeField,wpobjHyperlink]		
ClipboardOptions	[]		
ColorDesktop	clBtnShadow		
Constraints	(TSizeConstraints)		
Ctl3D	True		
Cursor	crIBeam		
DefaultIOFormat			
DragCursor	crDrag		
DragMode	dmManual		
EditBoxModes	[]		
EditOptions	[wpTableResizing,wpTableColumnResizing]		
EditOptionsEx	[wpKeepCellsWhenCombiningCells]		
Enabled	True		
FieldObjectTextAttr	(TCharacterAttr)		
FormatOptions	[wpIgnoreKeep,wpIgnoreKeepN,wpDisab]		
GraphicPopupMenu			
Header	(TTextHeader)		
Height	363		
HelpContext	0		
HelpKeyword			
HelpType	htContext		
HiddenTextAttr	(TCharacterAttr)		
Hint			
HyperLinkCursor	crHandPoint		
HyperLinkTextAttr	(TCharacterAttrTags)		
InsertPointAttr	(TCharacterAttrTags)		
LayoutMode	wplayFullLayout		
Left	28		
Name	WPRichText1		
OneClickHyperlink	False		
PageColumns	1		
PaperColor	clWindow		
ParentColor	True		
ParentShowHint	True		
PopupMenu	DemoPopupMenu1		
PrintParameter	(TWPPrintParameter)		
ProtectedProp	[ppParProtected,ppBookmark,ppProtect]		
ProtectedTextAttr	(TCharacterAttr)		
ReadOnly	False		
Resizing	100		
RTFText	16KB WPTools-Text [V5.17.2]		
RTFVariables	(TWPRTFExtraDataCollection)		
ScrollBars	ssBoth		
ShowHint	True		
SPANObjectTextAttr	(TCharacterAttrTags)		
SpellCheckStrategie	wpspCheckInInit		
TabOrder	0		
TabStop	False		
Tag	0		
TextObjectCursor	crHandPoint		
Top	28		
Transparent	False		
ViewOptions	[wpTraditionalMisspellers]		
Visible	True		
VRuler	WPVertRuler1		
WantReturns	True		
WantTabs	True		
Width	744		
WordWrap	False		
WPGutter	WPGutter1		
WPRuler	WPRuler1		
WPToolBar			
WriteObjectMode	wobRTF		
XBetween	144		
XOffset	144		
YBetween	144		
YOffset	144		
Zooming	100		

AcceptFiles / AcceptFilesOptions
activate graphic drag&drop functionality

attach actions

modify the way merged text is displayed

This format (RTF, ANSI, WPTOOLS, HTML) will be used by the load and save dialog

EditBoxModes: for auto sizing editors
EditOptions: Modify Undo/Redo functionality, possibility to change row height or column width and graphics.

FormatOptions
modify the display of tables (it is recommended to disable autosize tables), switch off keep support, display bookmark tags.

modify the way merge fields are displayed or hide them.

LayoutMode
normal layout, page layout, thumbnails - also see 'AutoZoom', 'Zooming'. To use an editor as preview control disable editing features in ViewOptions, EditOptions and property 'ReadOnly'.

ProtectedProp
to protect against editing mailmerge fields, text marked to be protected or to activate the **editfield** feature.

Document property variables, such as 'Author', 'Keywords'

ViewOptions:
Show special characters such as CR, tabs. Hide selection, show table gridlines.

attach GUI controls, actions, toolbar, ruler and gutter.

Loading and Saving

To load text into the editor You can use the functions LoadFromFile, LoadFromStream and also the properties AsString and SelectionAsString (the latter inserts the text). To move the cursor before the insertion or after the load modify the property CPosition.

To save the text You can use the functions SaveToFile, SaveToStream and AsANSISString. The function AsANSISString is useful to convert the text into a data string of a given format, i.e. AsANSISString('HTML') will create a string with HTML tags.

If the load and save functions ask for a second string parameter this is a "format string" - please read more here <http://www.wpcubed.com/manuals/formatstrings.htm>

In our [web based forum](#) we have posted several FAQ topics and articles.

The article "[Compose an e-mail in HTML format / Save image to HTML](#)" shows how to create a HTML e-mail.

The WPRichText component implements several functions which open a dialog box: Load, Save, SaveAs, InsertGraphicDialog and Insert.

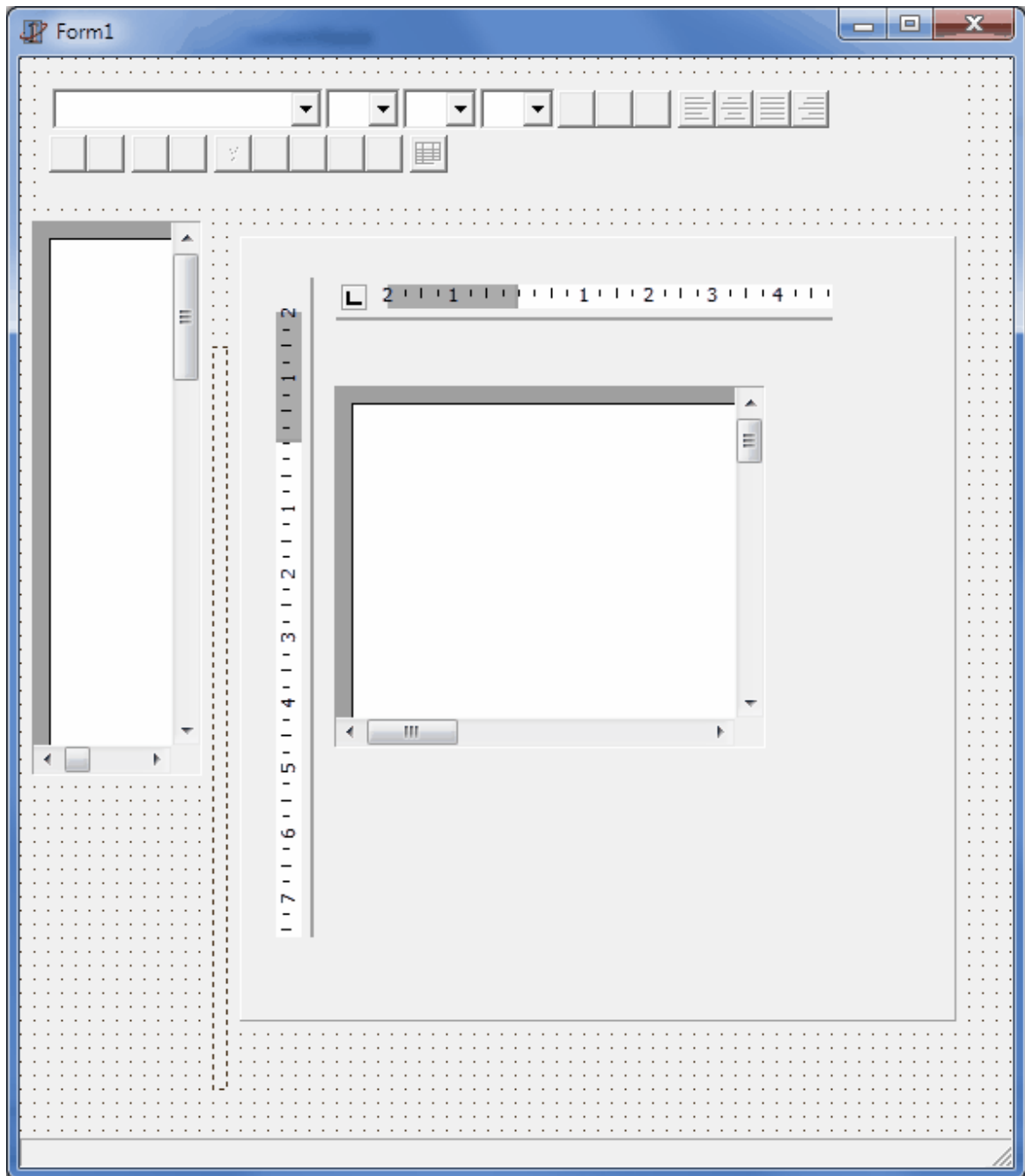
7.1 A) Mini Editor with Delphi 5

Although many of the new features can only be used with a modern edition of Delphi or C++ Builder we took some effort to still support Delphi 5. You will need the WPTools PRO edition with 100% source if you depend on Delphi 5.

This will empower you to offer your users quite some impressive features - despite the older compiler and VCL.

Hint: The powerful architecture of WPTools makes it easy to enhance this demo application to work with different documents which can be switched, just like a MDI application. We describe the technique in chapter "[Simulated MDI \(one editor, multiple documents\)](#)".

For our Delphi 5 Mini demo we just dropped a TWPRichText, 2 rulers on a TPanel, and a TWPPreview and TWPToolbar on a form. This looks like this:



Of course we could set the properties in the object inspector, but we do it in code to make clear what we needed to change.

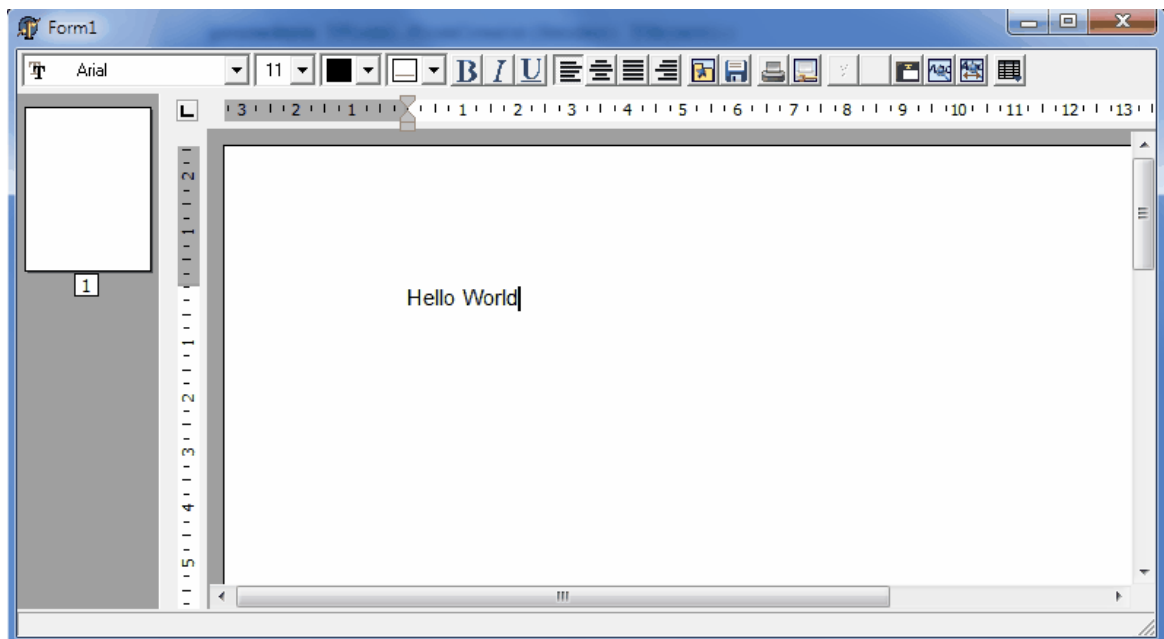
```
procedure TForm1.FormCreate(Sender: TObject);
begin
    // 1) Basis
    WPPreview1.Align := alLeft;
    Splitter1.Align:=alLeft;
    WPToolbar1.Align := alTop;
    Panel1.Align := alClient;
    Panel1.BevelInner := bvNone;
    Panel1.BevelOuter := bvNone;
    WPRuler1.Align := alTop;
```

```

WPVertRuler1.Align := alLeft;
WPRichText1.Align := alClient;
WPRuler1.Options := WPRuler1.Options - [wpNoVertRulerAttached];
// 2) connect TWPRichText
WPRichText1.WPRuler := WPRuler1;
WPRichText1.VRuler := WPVertRuler1;
WPRichText1.WPToolBar := WPToolBar1;
WPPreview1.WPRichText := WPRichText1;
WPPreview1.Configuration := wpPreviewThumbnails;
end;

```

When we start the application we see the classic WPTools editor (EXE Size in this state is 1972 KB):



We need to add the unit `WPRTDefConsts` to the uses clause. Now we can select the modern WPTools 7 interface style:

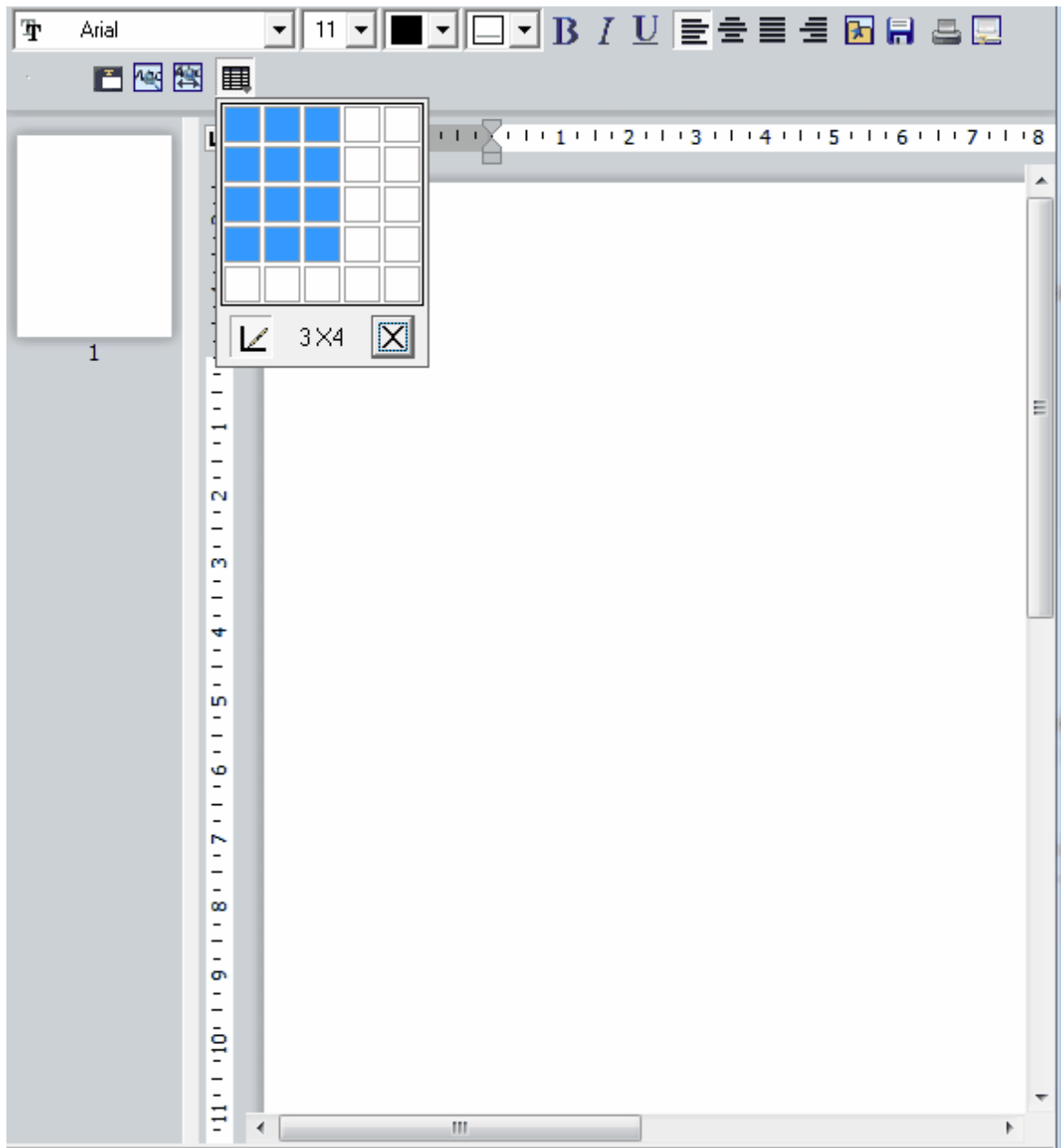
```

...
// 3) further configuration
WPToolBar1.DrawOptions := [ wptDrawPageShade ];
WPToolBar1.FlatButtons := true;
WPToolBar1.MarginBottom := 8;
WPToolBar1.BevelLines := [ wplBottomShade ];

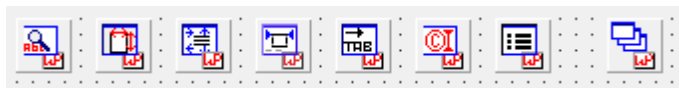
WPRuler1.DrawOptions := WPRuler1.DrawOptions + [wpDrawThemedBackground, wpDrawThemedRuler];
WPVertRuler1.DrawOptions := WPVertRuler1.DrawOptions + [wpDrawFramelines];
WPRuler1.DrawOptions := [wpDrawPageShade];
WPVertRuler1.DrawOptions := [wpDrawPageShade];
WPRichText1.ViewOptionsEx := WPRichText1.ViewOptionsEx + [wpPaintPageShade, wpPaintPageShade];
WPPreview1.ViewOptionsEx := WPPreview1.ViewOptionsEx + [ wpPaintPageShade ];

```

You will see the improvement immediately:



None of the dialogs do work yet. We need to drop some more components:



In the last component, the new `TWPDialogCollection` component, we assign the single dialog instances at designtime. The property `WPRichText.WPDialogCollection` we change to "WPDialogCollection1".

WPDIALOGCOLLECTION1: TWPDIALOGCOLLECTION	
BorderDialog	WPParagraphBorderDlgEx1
BulletAndOutlineDialog	
FormulaDialog	
InsertSymbol	WPSymbolDlgEx1
ManageHeaderFooter	
Name	WPDIALOGCOLLECTION1
PageDialog	WPPagePropDlg1
ParagraphDialog	WPParagraphPropDlg1
PreviewDialog	
ReportBandsDialog	
SaveAsPDF	
SearchReplace	WPSearchReplaceDlg1
SpellCheck	
SpellCheckConfigure	
StyleDialog	
StylesheetDialog	
TabDialog	WPTabDlg1
TableDialog	

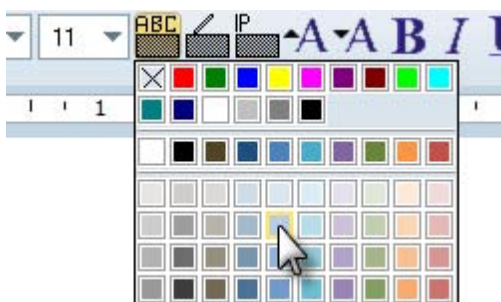
Now we can add popup dialog to make it possible to configure the toolbar. The popup is assigned to TWPToolBar. We need to add WPTBarConfig to the uses clause.

To show the configuration dialog use this code:

```
procedure TForm1.Configure1Click(Sender: TObject);
begin
  WPToolbarConfigurate( WPToolbar1, Self, 'Configure Toolbar' );
end;
```

The configuration is stored in the property ConfigString. If this property is not empty, ie. ''; the properties sel_ActionIcons etc. will be ignored. Please see the XE3 demo in the same folder to learn more about this feature.

You can activate the new color drop down elements with the property **StandardColorDropdowns** of the TWPToolbar set to false. The design colors are loaded from the string array WPDesignColors[0..9] (from WPRTEPlatform).



The EXE size of the demo application is now 2255 KB. It includes already the most important dialogs, the editor, RTF and HTML loading and saving etc.

Note: In case you get an error message when a form should be displayed you will need to open the respective unit and let Delphi ignore the problem. Since the Delphi 5 VCL misses some modern properties, we decided to not use it as lowest common

denominator.

In case you want to use the old style toolbar icons You can add the compiler conditional `OLDTBRESOURCE` and do a BuildAll:



7.2 B) Localization - change Language in Dialogs

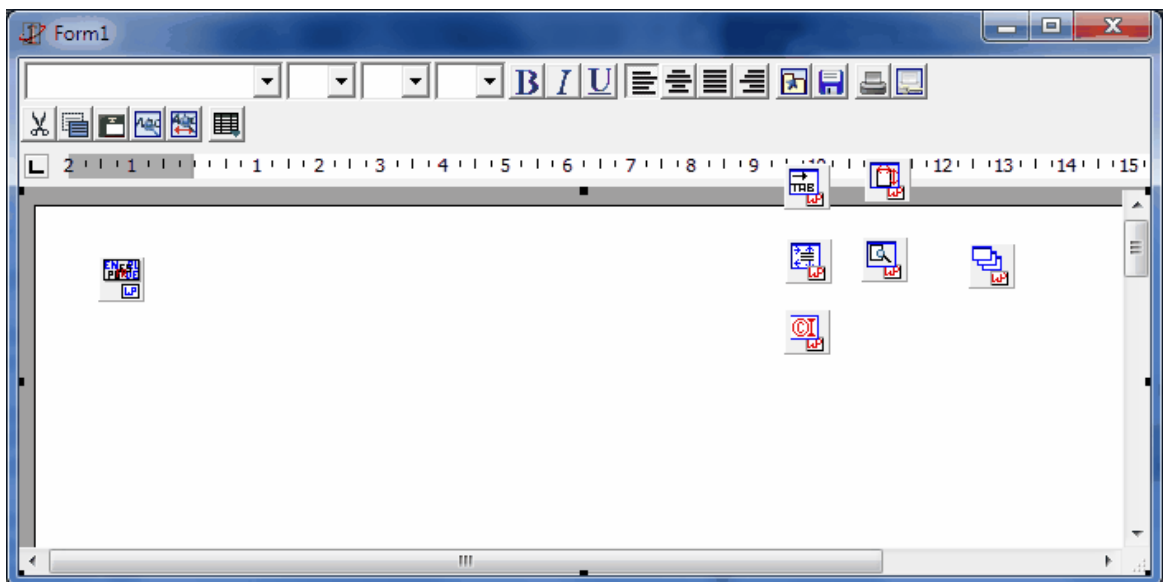
For anyone wanting to work with inch units, please add the following line in your code (for example, in the FormCreate event).

```
GlobalValueUnit := euInch;
```

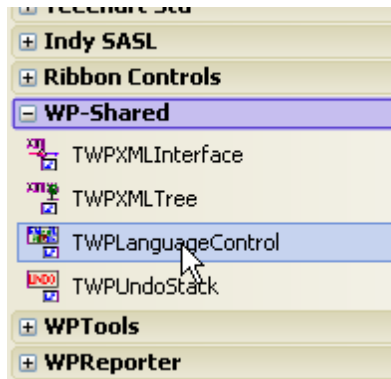
WPTools Version 7 supports the localization of the texts which are used for filters and error messages and also the localization of the provided property dialogs.

Steps to update the dialog language in WPTools 5, 6 and 7:

We start with a simple form:



Now we add the component TWPLanguageControl to the form



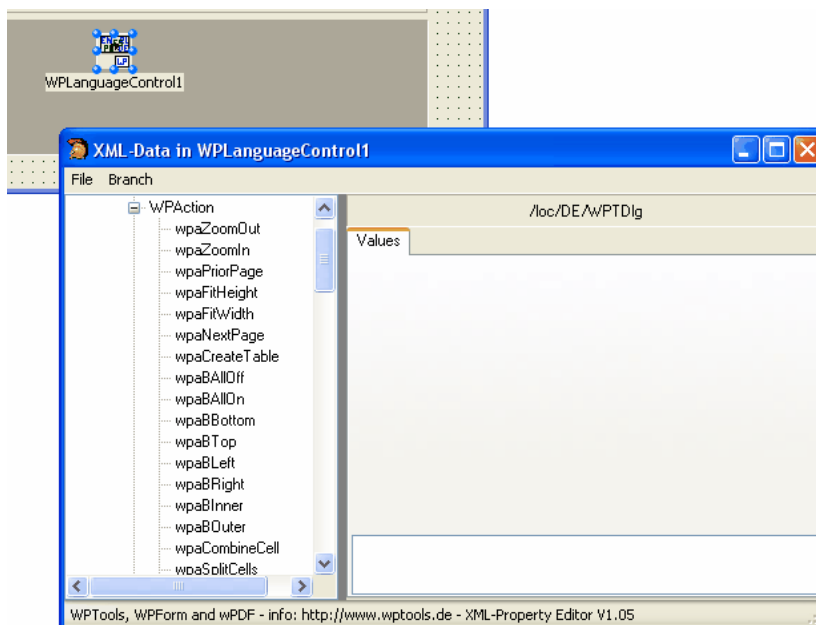
A double click opens the XML editor.

Here we can load the language file WPLocalization_2013.xml which is provided under **Demos \ B) _Localization \ XMLSources**.

Close the form with File/OK.

If you need to update a project you can also merge(!) in the new XML data which was added in WPTools 6 or WPTools 7.

If you need to translate the XML please save the EN branch into a different file. Now you can use an editor such as Notepad++ to edit that file. You need to rename `<EN>...</EN>` to a different language shortcut.



Please note that the properties **Active**, **AutoLoadString** and **AutoSaveStrings** are **not supported** in WPTools 5, 6 or WPTools 7.

Here we need to create a COM interface like this:

Add the units **WPUtil**, **WPActnStr** to the uses clause.

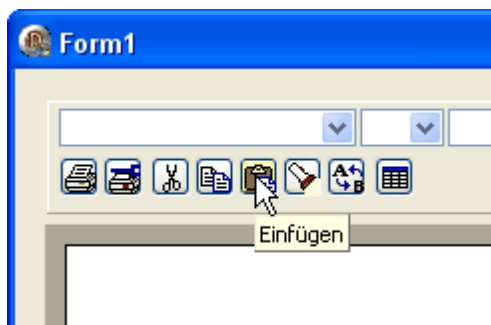
To active use this code in the **OnCreate** event of the form:

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    WPLangInterface := TWPLocalizationInterface.Create(WPLanguageControl1);  
    WPLanguageControl1.GlobalLanguage := 'DE';  
    WPLocalizeLoadForms := TRUE;  
    WPTools_LoadVCLStrings;  
    WPTools_LoadActionStrings;  
    WPToolbar1.ShowHint := true;  
end;
```

In the **OnDestroy** event add this:

```
procedure TForm1.FormDestroy(Sender: TObject);  
begin  
    WPLangInterface.Free;  
end;
```

Done: We have hints in German:



And Dialogs, too:



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
```

You need to create the **TWPLanguageControl** and the Interface on the Form which is created first in the project.

If necessary You can put it into a **Datamodul**.

How does the localization work?

In WPTools Version 7 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. }
end interface
```

```

    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 ancestor 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);`

7.3 C) Work with images

7.3.1 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 Version 7 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 FWPTTextObjs 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.

7.3.2 About Linked Images

Linked images use the string property StreamName to identify their contents. If this property is not an empty string, the binary image data will not be saved in a document format which otherwise allows embedding, such as RTF and WPT. To HTML images can only be saved if a "StreamName" is provided.

The event **OnHTTPRequestImage** is used to load the image data for images placed in HTML code **and** also for linked images loaded in RTF or WPT format. This event should be used to load images from a database which are only referenced in the text blob.

Please read the FAQ at

<http://wpcubed.com/forum/viewtopic.php?p=3287#3287>

The event **OnPrepareImageforSaving** can be used to create image files (or database records) for images which are embedded.

This example code will create new files for all images which are not embedded. They will also be compressed.

```
procedure TForm1.WPRichText1PrepareImageforSaving(
  RTFData: TWPRTFDataCollectionBase; Writer: TWPCustomTextWriter;
  TextObject: TWPTextObj; var DontSave: Boolean);
begin
  if ConvertEmebddedtoLinked.Checked then
    begin
      if TextObject.IsImage and (TextObject.ObjRef.StreamName='') then
        begin
          TextObject.ObjRef.StreamName :=
            ExtractFileName(
              TextObject.ObjRef.SaveToFile( Writer.SavePath, 'Test', '%d',
            );
        end;
      end;
    end;
end;
```

This example handler for **OnHTTPRequestImage** is used to load images through Indy.

```
procedure TForm1.WPRichText1RequestHTTPImage(RTFData:
  TWPRTFDataCollection;
  Reader: TWPCustomTextReader; const LoadPath, URL: String;
  TextObject: TWPTextObj; var Ok: Boolean);
var stream : TMemoryStream;
    loadurl : string;
begin
```

```

if pos('http:', lowercase(URL)) > 0 then
    loadurl := URL
else loadurl := LoadPath + URL;

stream := TMemoryStream.Create;
try
    Panel2.Caption := URL;
    try
        Application.ProcessMessages;
        // this example uses INDY
        IdHTTP1.Get(loadurl, stream);
    except
        on e : Exception do Panel2.Caption := URL + '-->' + e.
Message;
end;
if stream.Size > 0 then
    try
        TextObject.LoadObjFromStream(URL, stream);
    except
        on e : Exception do Panel2.Caption := URL + '-->' + e.
Message;
end;
ok := TRUE;
if ok then Panel2.Caption := '';
Application.ProcessMessages;
finally
    stream.Free;
end;
end;
end;

```

7.3.3 Example Code

1) InsertGraphicDialog

The easiest way to insert an image is to call the method `InsertGraphicDialog`. It has the optional parameters

```

filter: string = "";
InsertLink: Boolean = FALSE;
ObjectModes: TWPTTextObjModes = [];
path : string = "

```

filter can be used to specify a file extension filter for the open dialog.

InsertLink = true will create a linked image. When a file is saved, such an image will not be embedded. Instead the filename will be saved and, at load time, provided to the event **OnRequestHTTPImage**.

A linked image has a non empty property `StreamName`.

ObjectModes contains a set of flags which will be used for the new image object. You can specify `wpobjRelativeToParagraph` or `wpobjRelativeToPage` to insert a movable image.

`path` is the optional initial director for the open dialog.

Example:

```
procedure TForm1.BtnInsertImageDialogClick(Sender: TObject);
begin
    WPRichText1.InsertGraphicDialog(
        'Image Files@*.bmp',
        false,
        [wpobjRelativeToParagraph],
        ExtractFilePath(Application.EXENAME));
end;
```

2) Insert a linked image, Alternative 1

We simply provide a name, i.e. "RED" which is used in the event OnRequestHTTPImage to load an image. This can be very useful in database applications to store the images outside of the text blobs to improve performance.

```
procedure TForm1.InsertLinkedImageClick(Sender: TObject);
begin
    WPRichText1.InsertGraphic('RED', true, []);
end;

procedure TForm1.WPRichText1RequestHTTPImage(
    RTFData: TWPRTFDataCollectionBase; Reader: TWPCustomTextReader;
    const LoadPath, url: String; TextObject: TWPTTextObj; var Ok: Boolean);
begin
    TextObject.LoadObjFromFile(
        ExtractFilePath(Application.EXENAME) + url + '.bmp');
    Ok := true;
end;
```

3) Insert a linked image - Alternative 2 for delayed loading

Instead of loading the image in It is also possible use the event OnTextObjectPaint for this. This has the advantage that the image can be loaded later and only if it is visible. It is also possible to load alternative, smaller versions of the same image.

```
procedure TForm1.InsertImageForOnPaintClick(Sender: TObject);
begin
    WPRichText1.TextObjects.InsertNewObject(wpobjImage, 'GREEN');
    WPRichText1.ReformatAll(false, true);
end;

procedure TForm1.WPRichText1TextObjectPaint(Sender: TObject;
    pobj: TWPTTextObj; toCanvas: TCanvas; XRes, YRes, X, Y, W, H,
    BASE: Integer; PageRef: TWPVirtPage; Modes: TWPTTextObjectPaintModes;
    const CanvasExtraAttr: TWPPaintExtraParams;
    var ContinueMode: TWPTTextObjectPaintResult);
var s : String;
begin
    if (pobj.ObjType=wpobjImage) and (pobj.ObjRef=nil) then
        begin
            s := ExtractFilePath(Application.EXENAME) + pobj.Source + '.bmp';
            if (s<>'') and FileExists(s) then
```

```

    begin
      pobj.LoadObjFromFile(s);
      pobj.ObjRef.StreamName := pobj.Source; // MAKES it "LINKED"
      pobj.GetWHFromContents(1);
    end;
    ContinueMode := ContinueMode - [wpobjPaintRedCross];
  end;
end;

```

It is required to set the property **ObjRef.StreamName** to avoid embedding the image data when the text is saved.

4) Insert a graphic as object:

```

procedure TForm1.InsertGraphicClick(Sender: TObject);
var txtobj: TWPTextObj;
begin
  if OpenFileDialog2.Execute and (WPRichText1.ActiveParagraph <> nil)
  then
    begin
      WPRichText1.SetFocus;
      txtobj := WPRichText1.Memo.RTFData.TextObjects.Insert(
        WPLoadObjectFromFile(
          WPRichText1.Memo.RTFData,
          OpenFileDialog2.FileName), 1440, 1440);
      // Code to change the graphic, for example change width and
      height of the 'PositionMode'
      if txtobj <> nil then
        begin

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

```

Note: If you need to work directly with a TParagraph you can use the method **TParagraph.AppendNewObject** to create a new TWPTextObj object. To the ObjRef property of this object you can assign the TWPObj instance created by WPLoadObjectFromFile. (See [CreateTable](#) demo)

4) Modify the currently selected object:

With **TextObjects.SelectedObj** You have access to the TWPTextObj which is the anchor of the current object.

```

procedure TForm1.ChangeObjectPositionAndWrapMode(Sender:
TObject);
begin
  if WPRichText1.TextObjects.SelectedObj<>nil then
    begin

```

```

        WPRichText1.TextObjects.ChangePositionMode(
            WPRichText1.TextObjects.SelectedObj,
            wpotPar, wpwrBoth);
    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.

5) 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;

```

6) Search and Replace text with 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;
end;

```

Alternative, using a graphic from an TImage object:

```

WPRichText1.Finder.ToStart;
while WPRichText1.Finder.Next('[sig]') do
    begin
        WPRichText1.CPPosition := WPRichText1.Finder.FoundPosition;
        WPRichText1.Finder.FoundText := '';
    end;

```

```

    WPRichText1.TextObjects.InsertCopy(Image1.Picture.Graphic);
end;
WPRichText1.DelayedReformat;

```

7.3.4 Provide a Graphic Popup Menu

Please insert this popup menu into 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
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;
end;

```

7.3.5 Images and Mailmerge

This code will insert a mail merge field:

```
procedure TForm1.InsertImageFieldClick(Sender: TObject);
begin
    WPRichText1.InputMergeField('IMAGEFIELD', '[IMAGE]');
end;
```

This code launches the merge process

```
procedure TForm1.MergeTextClick(Sender: TObject);
begin
    WPRichText1.MergeText;
end;
```

The event handler for `inspname` inserts the image or replaces an existing image with different data.

```
procedure TForm1.WPRichText1MailMergeGetText(Sender: TObject;
    const inspname: String; Contents: TWPMInsertTextContents);
begin
    if Contents.StartInspObject.Source <> 'GREEN' then
    begin
        Contents.LoadImageFromFile(
            ExtractFilePath(Application.ExeName) +
            'green.bmp', -1, -1);
        Contents.StartInspObject.Source := 'GREEN';
    end
    else
    begin
        Contents.LoadImageFromFile(
            ExtractFilePath(Application.ExeName) +
            'red.bmp', -1, -1);
        Contents.StartInspObject.Source := 'RED';
    end;
end;
```

We use `Contents.LoadImageFromFile` to load and update the image. The Parameters **w** and **h** are set to -1 to make it use the default size (=physical) or the current width and height.

The **Source** parameter of the image is used to remember which file was loaded. This makes it possible to toggle between two states.

7.4 D) How to use the "Default Editor"

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

Yes, with WPTools Version 7 you can.

WPTools 7 includes the DefaultEditor from WPTools 6 (WPDefEditor) and also the new WPDefEditor7 which is using the DefaultActions7. They include new image

lists and a different menu. The new action datamodule is incompatible to Delphi 5. We use that in the "Ribbon" demo.

7.4.1 Using "Old" module

A) The is the complete program code in Delphi:

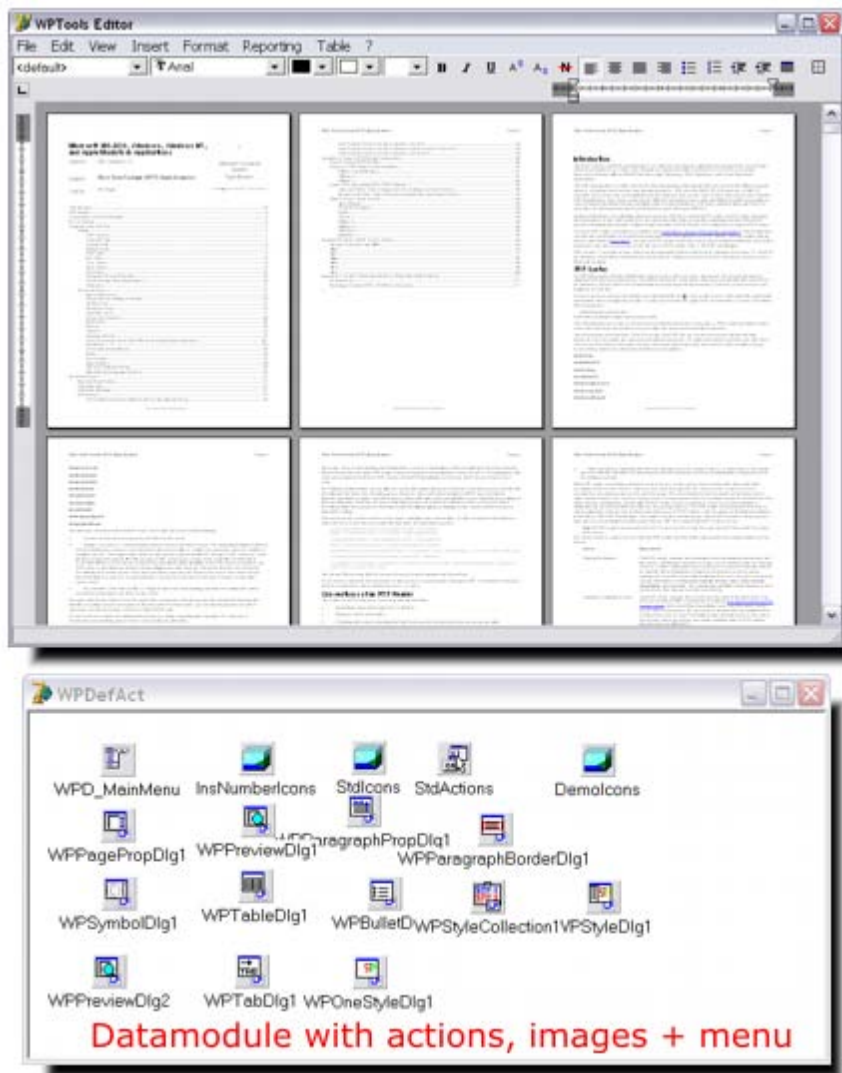
```
program UseDefEditor;  
  
uses  
    Forms, WPDefEditor;  
  
{$R *.res}  
  
begin  
    Application.Initialize;  
    Application.CreateForm(TWPToolsEditor,WPToolsEditor);  
    Application.Run;  
end.
```

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.

You can also use the default editor within a MDI project:

// This event for auto free !

```
procedure TWPMdiDemo.WPDefEditorFormClose(Sender: TObject; var Action: TCAction);
begin
    Action := caFree;
end;
```

// Create a new wptools default editor as MDI child

```
procedure TWPMdiDemo.NewEditorClick(Sender: TObject);
var
    lForm : TWPToolsEditor;
begin
```

```
WPisMDIApp := true; // Make the Toolbar MDI compatible
lForm := TWPToolsEditor.Create(nil);
lForm.FormStyle := fsMDIChild;
lForm.OnClose := WPDefEditorFormClose;
end;
```

7.4.2 Using V7 Module

Instead of WPDevActions you will need to use WPDevActions7.

7.5 E) Ribbon Applications

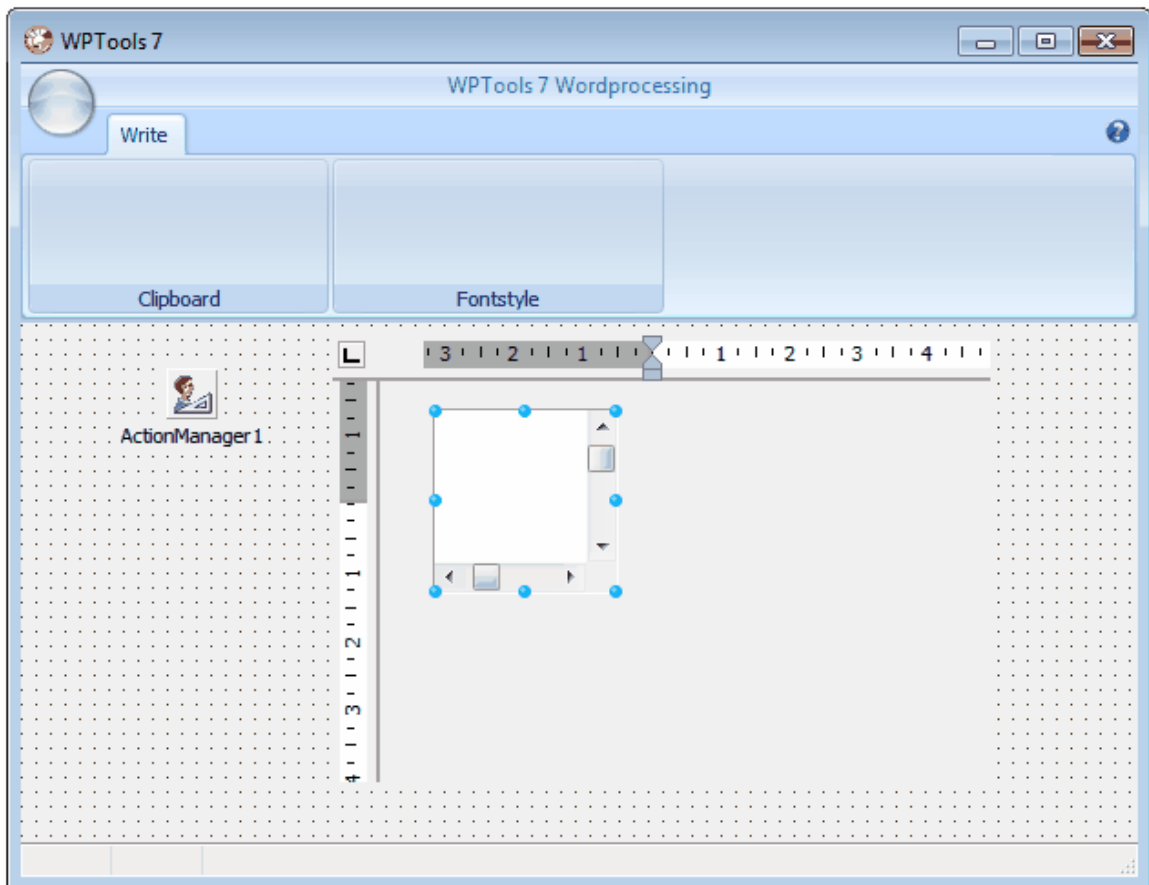
7.5.1 Standard XE3 Ribbons

Our XE3 Ribbon demo uses the default action data module **wpDefActions7** which must be added to the project.

Only then the actions are visible to the IDE. If **wpDefActions7** was not added or not found, opening the project in the IDE will remove all links to the actions!

For a ribbon application you need a TRibbon object on your form. You can also place a TWPRichText, TWPRuler directly on the form or inside of a TPaint. If you place it into a TPanel, you can later use a splitter to offer a splitted GUI with an editor and a thumbnail view.

You need an ActionManager to configure the the ribbon.



Attach `WPDefAct.StdIcons` and `WPDefAct.StdActions` to the `ActionManager`. Then you can double click on the `ActionManager` and configure the ribbon by using drag and drop.

You need to add this units to the uses clause:

```
// manually added:
, WPRTEPlatform, WPRTEDefsConsts, WPRTEPaint, WPRTEEdit, WPRTEFormatA, WP
// Default Actions
, wpDefActions7.
```

You need to manually add a procedure to the form:

```
procedure TWPTEditor.DoGetWPRichText(Sender: TObject; var wp: TWPCustomRichTextEditor);
begin
    wp := WPRichText1;
end;
```

In `OnCreate` you need some code for initialization:

```
procedure TWPTEditor.FormCreate(Sender: TObject);
begin
    EditPanel.Align := alClient;
    WPRichText1.Align := alClient;
    WPActions := TWPDefAct.Create(Self);
```

```

WPRichText1.WPDialogCollection := WPActions.WPDialogCollection1;
WPActions.OnGetWPRichText := DoGetWPRichText;

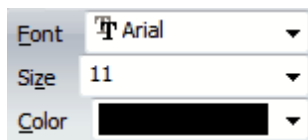
// Here we update the link to the insnace of the datamodule
with ActionManager1.LinkedActionLists[0] as TActionListItem do
begin
    ActionList := WPActions.StdActions;
end;

// Configure TWPRichText
WPRichText1.EditOptions := WPRichText1.EditOptions
    + [wpActivateRedo, wpActivateRedoHotkey, wpActivateUndo, wpActivateUndo];

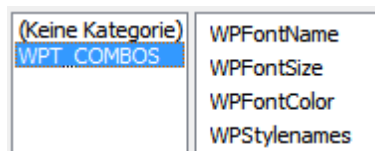
WPRichText1.ViewOptionsEx := [wpPaintThemedBackground];
WPRuler1.Options := WPRuler1.Options - [wpNoVertRulerAttached];
WPRuler1.DrawOptions := [wpDrawFramelines, wpDrawThemedBackground];
WPVertRuler1.DrawOptions := [wpDrawFramelines, wpDrawThemedBackground];
end;

```

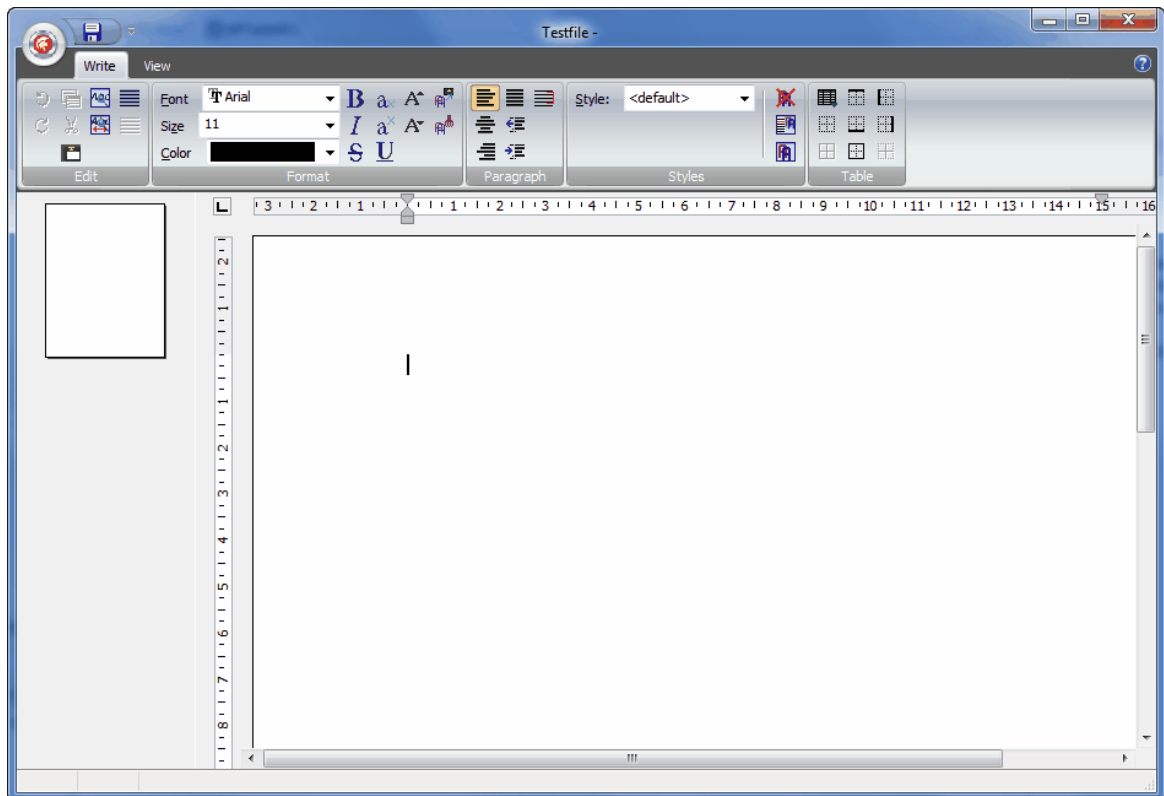
You can place TWPComboBox elements on the ribbon:



They are attached to the TWPRichText by a regular ActionList which is referenced by the TWPRichText's property ActionList. Inside the action list you need a TWPToolsCustomEditControlAction for each of the combos. The property AttachedControl must reference the combo, the property AttachedControlStyle is ignored.



After the ribbon was configured the application can look like this



General Hints:

If you get the message "class Actionlist not found" when starting the application simply place an empty TActionlist on the form.

If you get a GPF when loading a project in Delphi XE close Delphi and delete all redundant project files x.RES, x.LOCAL, x.DSK and reopen Delphi.

If you need to create a color drop down element, You can use the function **WPCreateColorForm** from unit WPColSel:

Example:

```
var col : TColor;
begin
  if WPCreateColorForm(Self, Sender as TSpeedButton, WPRichText1, col) then
    WPRichText1.CurrAttr.Color :=
      WPRichText1.CurrAttr.ColorToNr( col, true );
  WPRichText1.SetFocus;
end;
```

7.5.2 TMS Office 2010

In this chapter we describe how to use the TMS ribbon control to create a modern GUI for WPTools.

For more information please see [tmssoftware](http://tmssoftware.com).

Hint: The powerful architecture of WPTools makes it easy to enhance this demo application to work with different documents which can be switched, just like a MDI application.

We describe the technique in chapter "[Simulated MDI \(one editor, multiple documents\)](#)".

7.5.2.1 Start

Create a new Project



and save it under a new name.

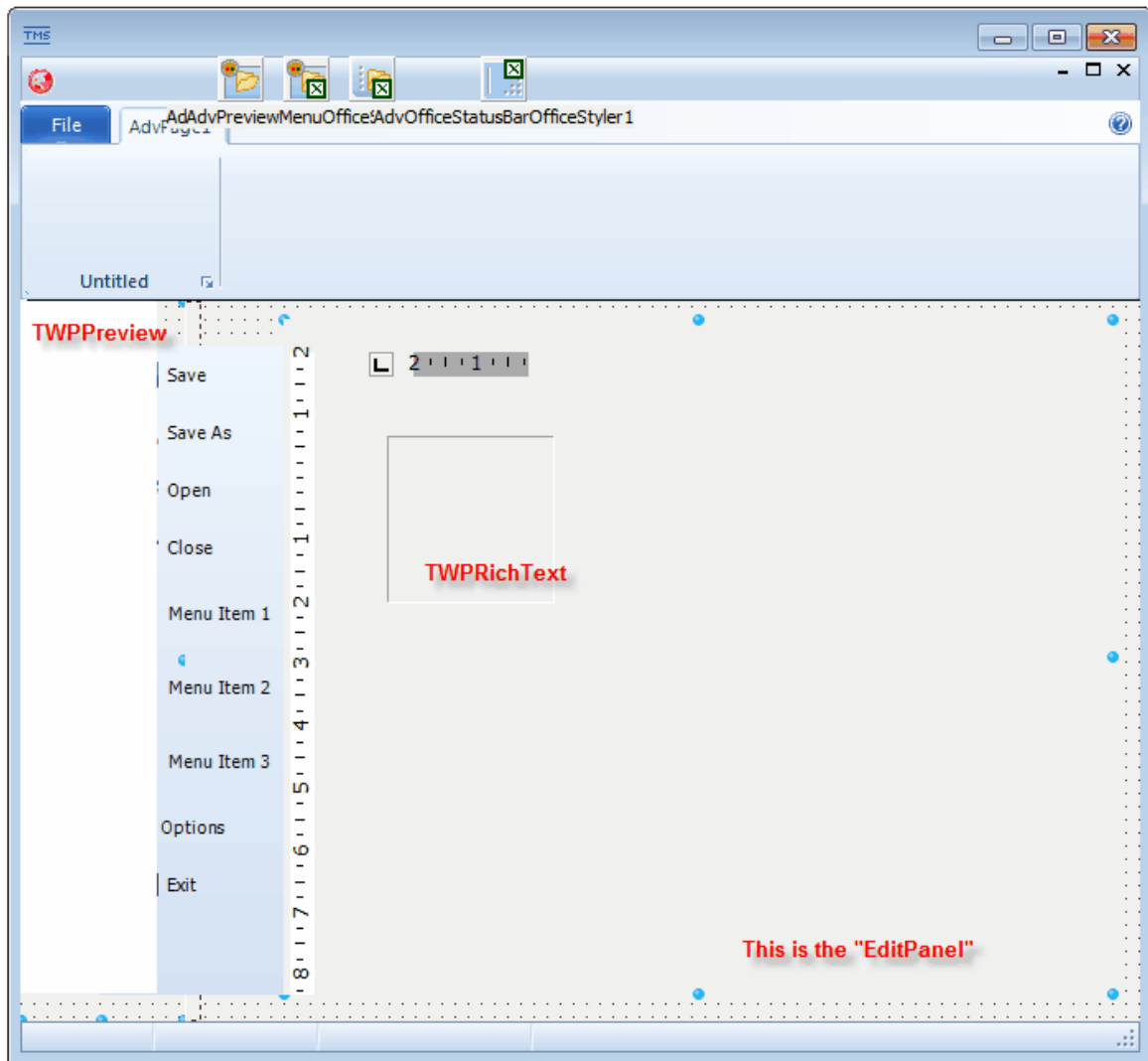
Add Unit wpDefActions7 to project and add it to the uses clause in the main unit.

Also add the units **WPRTEDefs**, **WPRTEPlatform**, **WPRTEPaint**, **WPRTEEdit**, **WPRTEFormatA** to the uses clause.

Now add a Panel, called "Edit Panel" to the form.

Inside this Panel please place a TWPRichText, a horizontal TWPRuler and a vertical TVerRuler.

On the left side of the EditPanel place a TWPPreview and a splitter, both with "Align = alLeft".



Now You can use the designer or some source code in OnCreate to connect and initialize the components.

```
procedure TWPEditor.FormCreate(Sender: TObject);
begin
    WPRichText1.WPRuler := WPRuler1;
    WPRichText1.VRuler := WPVertRuler1;
    WPPreview1.WPRichText := WPRichText1;
    WPPreview1.LayoutMode := wpThumbNailView;
    WPRuler1.Align := alTop;
    WPVertRuler1.Align := alLeft;
    WPRichText1.Align := alClient;
    // This is the parent of WPRichText1 and the rulers
    EditPanel.Align := alClient;
    ActiveControl := WPRichText1;
end;
```

Now you should be able to start the project and test the editing. If the editor does not work, the unit WPRTEFormatA was not added to the uses clause.

7.5.2.2 Populate Edit Menu

In step one we already added the unit WPDevActions7 to the project. This is necessary to let the delphi designer use it.

We now create references for the project at runtime:

a) This variable to reference the action data module has been already created in unit WPDevActions7:

```
var WPDefAct: TWPDefAct;
```

b) Create an event handler which is used to tell the actions which WPRichText should be used:

```
procedure TWPEditor.GetCurrentEditor(Sender: TObject; var wp: TWPCustomRichText);  
begin  
    wp := WPRichText1;  
end;
```

c) Create the datamodule and initialize in Form.OnCreate

```
procedure TWPEditor.FormCreate(Sender: TObject);  
begin  
    ....  
    WPDefaultActions := TWPDefaultActions.Create(Self);  
    WPDefaultActions.WPDefAct.OnGetWPRichText := GetCurrentEditor;  
    WPRichText1.WPDialogCollection := WPDefaultActions.WPDefAct.WPDialogCollection1;  
end;
```

Now it is possible to populate the first toolbar. We have set the property AutoArrangeButtons and AutoPositionControls to false.

On the first toolbar we place one AdvGlowMenuButton and 4 objects of type AdvGlowButton.

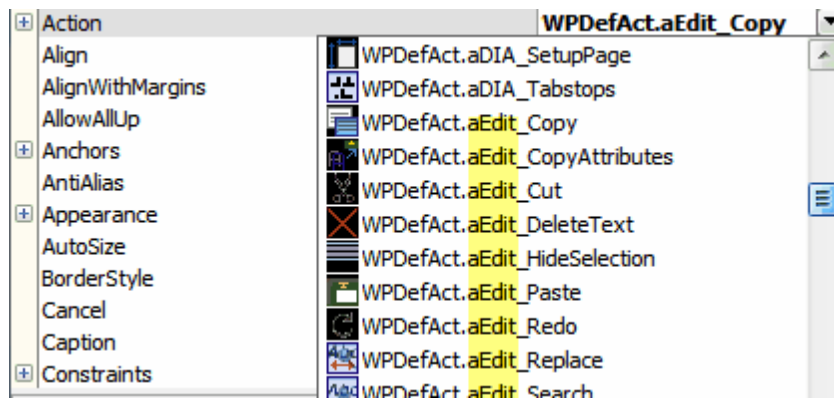


In the first button we use the property "Picture" to load a PNG image symbol. We set its property "Transparent" to true.

We create an OnClick event with this code. We can use the same handler in an PopupMenu to select the mode wpPasteAsText and wpPasteSimplified through the "tag" property 1 and 2.

```
procedure TWPEditor.PasteButtonClick(Sender: TObject);  
begin  
    WPRichText1.PasteFromClipboard( TWPPasteTextMode((Sender as TComponent).Tag) );  
end;
```

In the properties of the other 4 buttons we assign Actions from the standard action list:

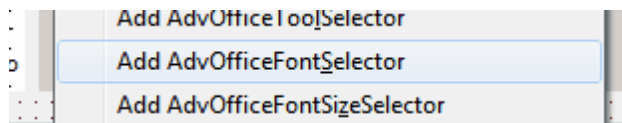


The actions are named with preceding categories to make it easier to locate them.
The property ShowCaption is set to FALSE.

7.5.2.3 Populate Style toolbar with Font and Color Selector

Now we use the special TMS comboboxes to work with TWPRichText.

Create a new toolbar and click right on it. Now you can select the AdvOfficeFontSelector:



Assigning the selected font to the editor is very simple. You only need an event handler like this.

```
procedure TWPEditor.AdvOfficeFontSelector1SelectFontName(Sender: TObject;
  AName: string);
begin
  WPRichText1.CurrAttr.FontName := AName;
  WPRichText1.SetFocus;
end;
```

Now we add a and a TAdvOfficeFontSizeSelector and 3 TAdvOfficeColorSelector for text color, highlight color and paragraph color.

This code is used to update the text:

```
procedure TWPEditor.AdvOfficeFontSizeSelector1SelectFontSize(Sender: TObject;
  ASize: Integer);
begin
  WPRichText1.CurrAttr.Size := ASize;
  WPRichText1.SetFocus;
end;
```

```
procedure TWPEditor.TextColorSelectorSelectColor(Sender: TObject;
  AColor: TColor);
begin
  WPRichText1.CurrAttr.Color := WPRichText1.CurrAttr.ColorToNr(AColor);
  WPRichText1.SetFocus;
end;
```

```

procedure TWPEditor.HighlightColorSelectorSelectColor(Sender: TObject;
  AColor: TColor);
begin
  WPRichText1.CurrAttr.BKColor := WPRichText1.CurrAttr.ColorToNr(AColor);
  WPRichText1.SetFocus;
end;

procedure TWPEditor.ParagraphColorSelectorSelectColor(Sender: TObject;
  AColor: TColor);
begin
  WPRichText1.CurrAttr.ParColor := WPRichText1.CurrAttr.ColorToNr(AColor);
  WPRichText1.SetFocus;
end;

```

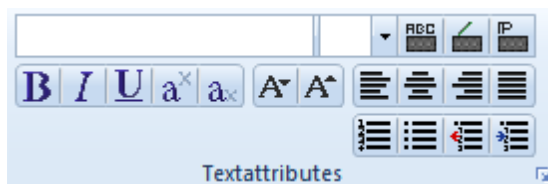
To update the controls according to the text attributes you need an event handler for the event **WPRichText.OnCharacterAttrChange**.

```

procedure TWPEditor.WPRichText1CharacterAttrChange(Sender: TObject;
  Attribute: TWPSetModeControl);
begin
  // Font Color
  AdvOfficeFontSelector1.ItemIndex :=
    AdvOfficeFontSelector1.Items.IndexOf(Attribute.FontName);
  // Font Size
  if Attribute.Size<=0 then
    AdvOfficeFontSizeSelector1.ItemIndex := -1
  else AdvOfficeFontSizeSelector1.SelectedFontSize := Trunc(Attribute.Size);
  // TextColor
  if Attribute.Color<=0 then
    TextColorSelector.SelectedColor := clNone
  else TextColorSelector.SelectedColor :=
    Attribute.NrToColor(Attribute.Color);
  // HighlightColor
  if Attribute.BKColor<=0 then
    HighlightColorSelector.SelectedColor := clNone
  else HighlightColorSelector.SelectedColor :=
    Attribute.NrToColor(Attribute.BKColor);
  // ParagraphColor
  if Attribute.ParColor<=0 then
    ParagraphColorSelector.SelectedColor := clNone
  else ParagraphColorSelector.SelectedColor :=
    Attribute.NrToColor(Attribute.ParColor);
end;

```

Now we can add additional buttons to change the writing mode and properties of the text.

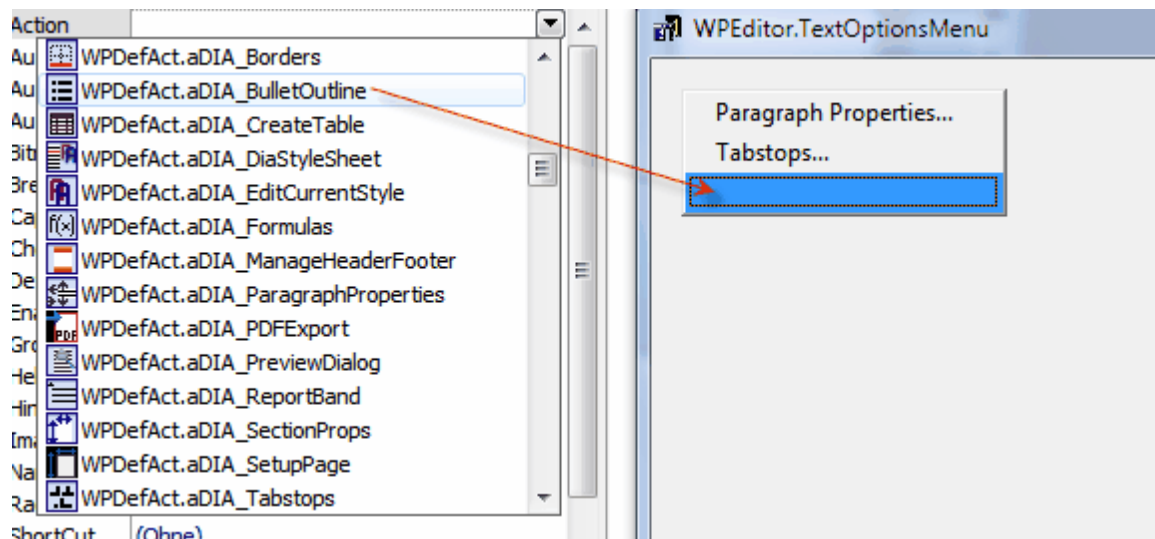


They all use actions from the WPToolDefaultAction datamodule.

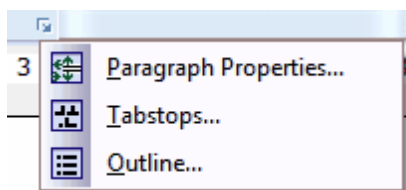
For the buttons we set the property *Rounded* to true and the property *Position* to bpLeft, bpMiddle ... and bpRight to create the grouped look.

All buttons which can also display a state (active/inactive) the property *Style* must be to bsCheck.

Now we create a popup menu (TAdvPopupMenu) and add actions to show important dialogs:



The menu is assigned to the property OptionsMenu of our style toolbar. Don't forget to assign WPDefAct.StdIcons to property Images of the popup menu.

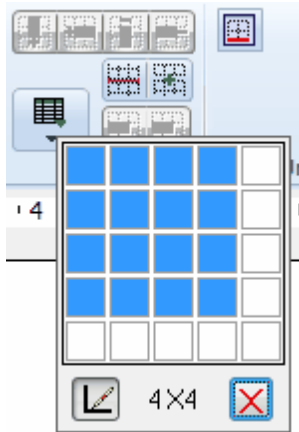


7.5.2.4 Populate Table toolbar

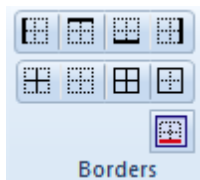
To create a table menu You can either use the standard action WPDefAct.aTbl_CreateTable or a TAdvOfficeTableSelector. If You use TAdvOfficeTableSelector, You need this code to create a table:

```
procedure TWPEditor.AdvOfficeTableSelector1SelectTableSize(Sender: TObject;
  Columns, Rows: Integer);
begin
  WPRichText1.TableAdd(Columns, Rows, [wptblActivateBorders]);
end;
```

If you use WPDefAct.aTbl_CreateTable You only need to place a button and assign the action.



We also add a button toolbar by using actions from `WPDefAct`. We set the property *Rounded* to true and the property *Position* to `bpLeft`, `bpMiddle ...` and `bpRight` to create the grouped look. All buttons which can also display a state (active/inactive) the property *Style* must be to `bsCheck`.



7.5.2.5 Make the File menu work

To do so we have to add the frame unit which implements the file menu. We renamed that to "FileMenuOverlay.pas".

The Menu is a `TAdvPolyMenu` and we can enter the property editor with a double click on the menu.

Here we can add some code to execute the default actions. By doing so, we avoid the dependency to a `TWPRichText` and make the approach more universal.

```
procedure TTMSFrame1.OpenBtnItemClick(Sender: TObject; Item: TCustomItem);
begin
    WPDefaultActions.WPDefAct.aFile_Open1.Execute;
end;

procedure TTMSFrame1.SaveAsBtnItemClick(Sender: TObject; Item: TCustomItem);
begin
    WPDefaultActions.WPDefAct.aFile_SaveAs.Execute;
end;
```

Using Actions here is also possible, but it makes it more difficult to show a different icon.

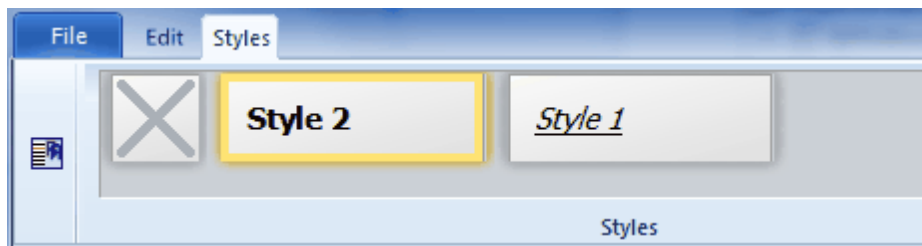
7.5.2.6 Style Scroller

We decided to separate the style menu from the edit menu, since this makes it possible to provide large previews of the styles. Since WPTools 7 includes a style scrolling component You only need to add a new page to the ribbon and place the `TWPStyleScroller`.

To attach it to the `TWPRichText` we need a `TActionList` "ActionList1" on the form. There we add a standard action of type `TWPToolsCustomEditControlAction`. Set its property *AttachedControl* to `WPStyleScroller1`.

The property *ActionList* of the `TWPRichText` must be set to `ActionList1`.

Now you can customize the `TWPStyleScroller`, pages can be drawn shaded or simple.



7.6 F) Mini Editor

With WPTools Version 7 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.

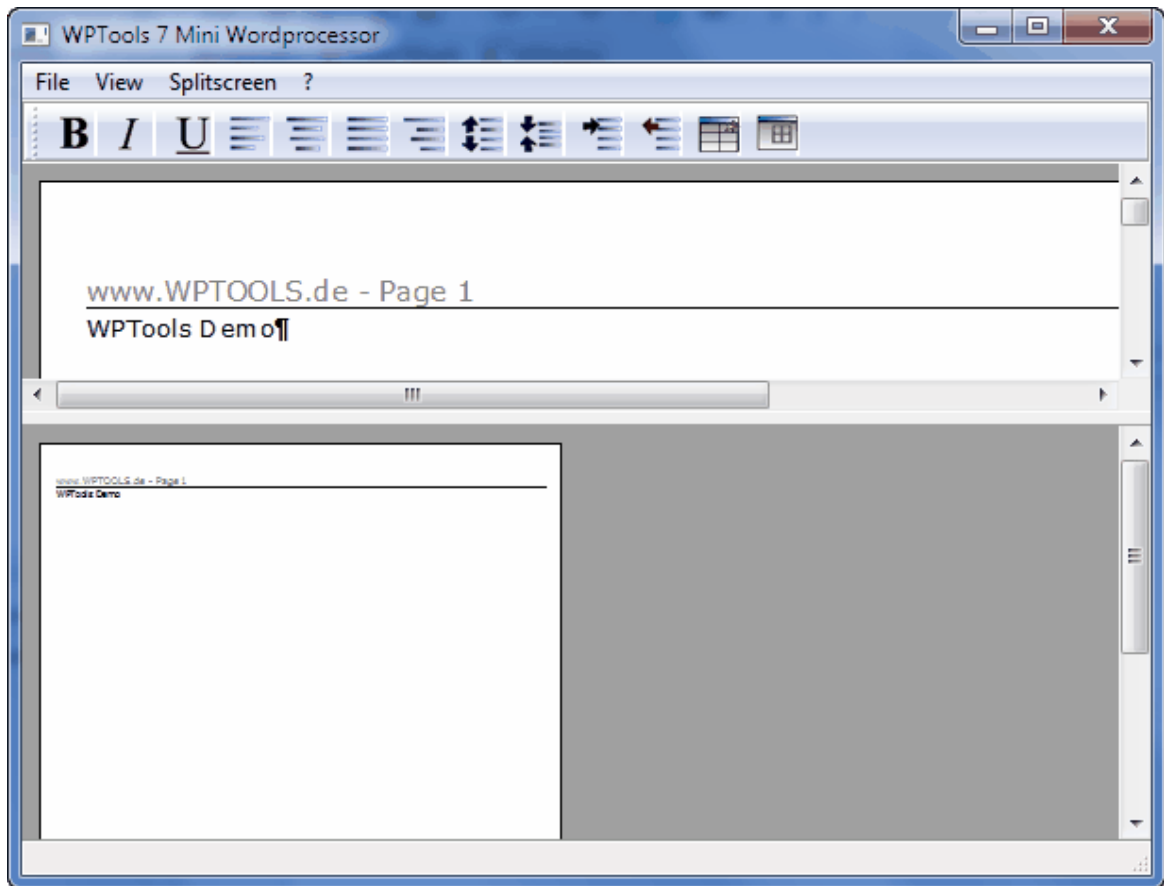
Our demo uses 4 variables defined inside the form interface:

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

You will need to manually add this unit references to "uses":

`WPRTEPlatform, WPRTEEdit`

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);
    RTFData.FormatOptions := [wpDisableAutoSizeTables]; //!!!!
    RTFDataProps := TWPRTFProps.Create;
    RTFData.RTFProps := RTFDataProps;

    WPRichText1 := TWPCustomRtfEdit.Create(Self);
    WPRichText1.Parent := EditPanel;
    WPRichText1.Align := alClient;
    WPRichText1.TabStop := FALSE;
    WPRichText1.AcceptFiles := TRUE;
    WPRichText1.Height := Height div 2;
    WPRichText1.Memo.SetRTFDataOrProps(RTFData, nil);
    WPRichText1.OnChangeCursorPos := DoChangeCursorPos;

    WPRichText1.AcceptFiles := true;
    WPRichText1.ViewOptions := [wpShowNL, wpShowCR];

    WPRichText2 := TWPCustomRtfEdit.Create(Self);
    WPRichText2.Memo.SetRTFDataOrProps(RTFData, nil);
    WPRichText2.Parent := EditPanel;
    WPRichText2.Align := alBottom;
```

```

WPRichText2.Height := Height div 2;
WPRichText2.OnChangeCursorPos := DoChangeCursorPos;

WPRichText2.AcceptFiles := true;
WPRichText2.ViewOptions := [wpShowNL];

```

If you want to also have a statusbar please place the editors inside of a TPanel.

```
WPRichText1.Parent := EditPanel;
```

We now assign the graphic popup menu. This menu is automatically used as context 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.

Note: To make the Mini Editor use the DOC import (based on the MS word converter DLLs which may or may not be installed on the system) you need to add the unit WPWordConv to the uses clause.

Hint: The TWPCustomRtfEdit can also be used in a thread save context, for example to create documents using mailmerge in a thread save context.

In this case call the constructor **TWPCustomRtfEdit.CreateDynamic** instead of TWPCustomRtfEdit.Create(nil) to create a dynamic text creation engine.

Please check out the demo "[ThreadSave](#)"

7.7 G) Low level text creation

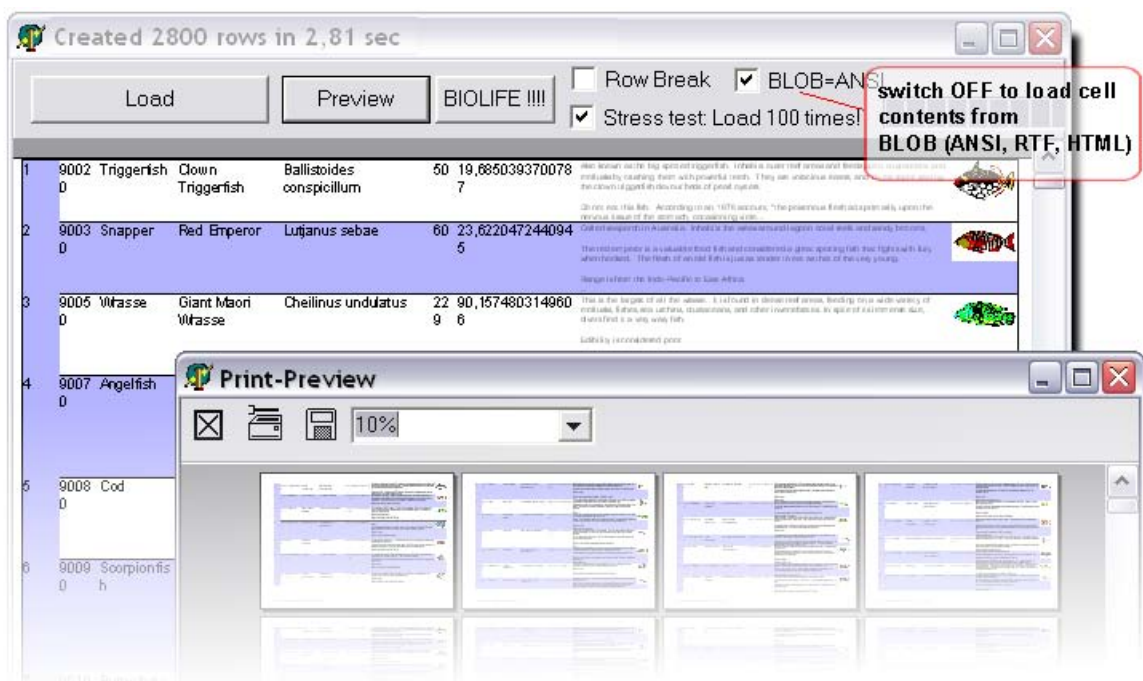
Geben Sie hier den Text ein.

7.7.1 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 clears 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: TWTextObj;
wpobj: TWPObj;
bit: TBitmap;
rowstyle: TWTableRowStyle;
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.ContentWidth div 3;
              obj.Height := wpobj.ContentHeight 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;

```

7.7.2 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 count 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.

Please see [reference](#) for supported WPAT_ codes.

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 TWPTTableCalc.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 lines
      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;
    end;
  end;

```

```

        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);par
            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;
end;

```

Note: The commands

```

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

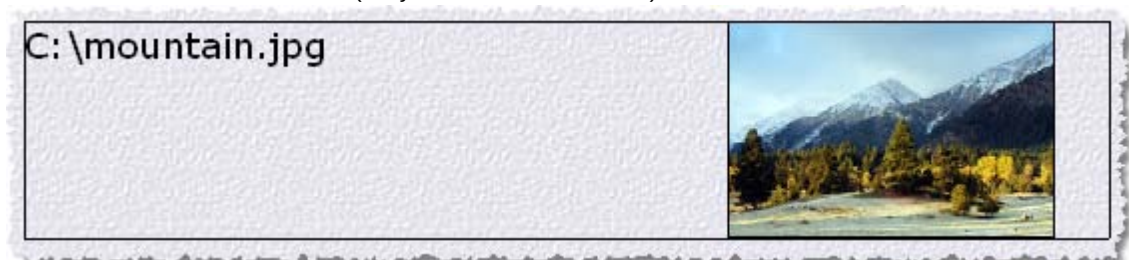
```

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: TWTextObj;
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);

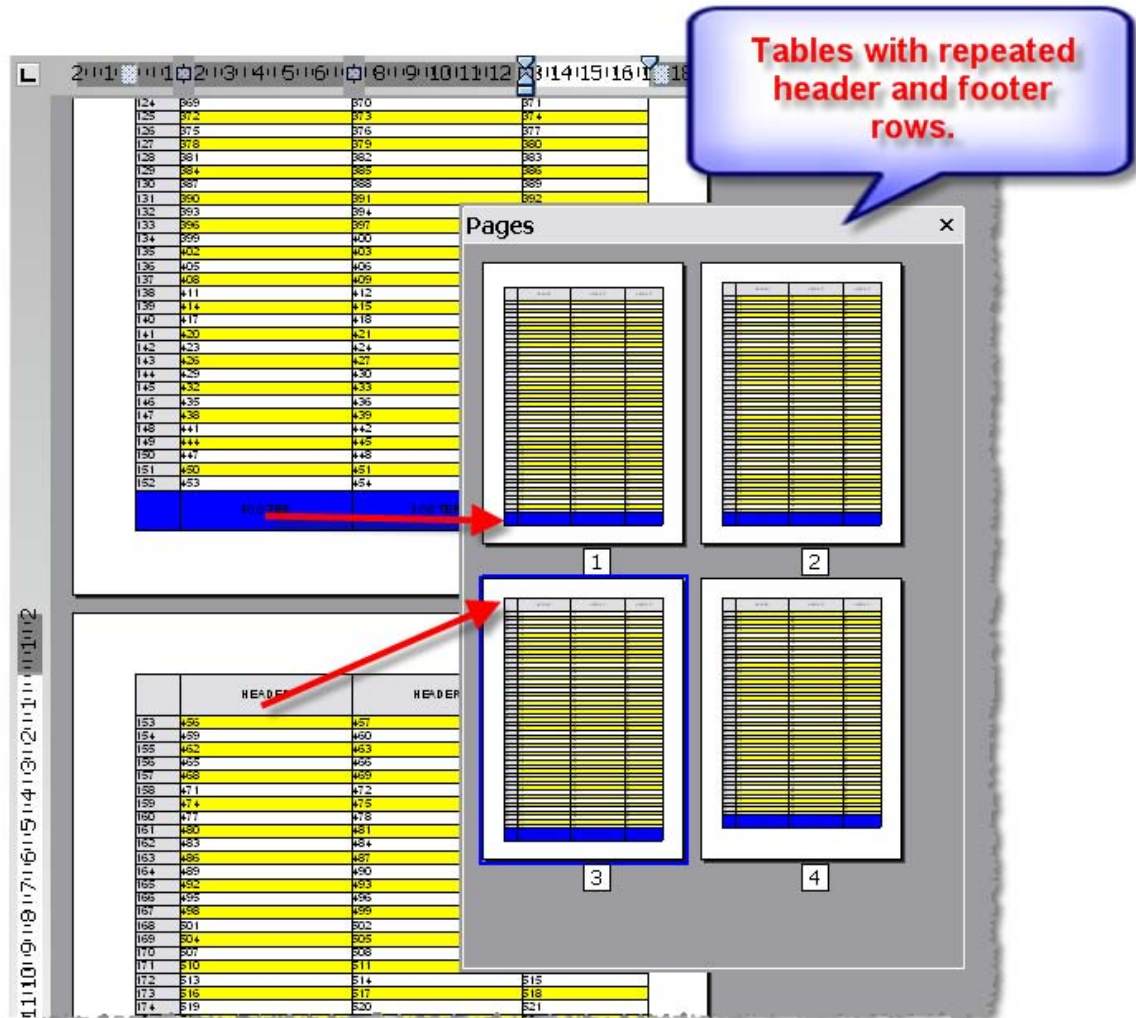
// Create the TWTextObj (which is the reference to image)
obj := tbl.Rows[rowoffset+0].Cols[1].AppendNewObject(wpobjImage, false, false, 0
);
obj.ObjRef := img;
obj.Frame := [wpframeFine];

// 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 `paprlsFooter` and `paprlsHeader` 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
  end
```

```

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;

```

7.7.3 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.

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);
```

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`!

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;

```

Append empty paragraph

```

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;
```

A paragraph with a green background:

This is 30 % green

```
// 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;

Please see [reference](#) for supported WPAT_ codes.

A paragraph with a red background and borders:

This is 50 % red, indented

```

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;
```

A paragraph with colored borders and spacing

Red, Green, Blue, Yellow

```

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;

```

A paragraph with tabstops

START fromto END

```

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](#).

A table with colored borders

Cell A	Cell B	Line 1 Line 2
second cell		

```

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;

```

7.7.4 Auto capitalisation

You can use the event `BeforeInitializePar` to update the current paragraph:

```

procedure TForm1.WPRichText1BeforeInitializePar(Sender: TObject;
  RTFEngine: TWPRTFEngineBasis; RTFDataBlock: TWPRTFDataBlock;
  par: TParagraph);
var pos_wstart, i : Integer; s : string;
begin
  // Current paragraph
  // Current paragraph
  if par = WPRichText1.ActivePar then
  begin
    i := WPRichText1.ActivePosInPar;
    // 1. Overread spaces to the left
    while (i>0) and par.IsSpace(i) do dec(i);

    // 2. Overread a word
    pos_wstart := -1;
    if not par.IsWordDelimiter(i) and (par.IsSpace(i+1)) then
    begin
      while (i>=0) and not par.IsWordDelimiter(i) do begin pos_wstart := i; dec(i); end;
      // 3. Overread spaces
      while (i>0) and par.IsSpace(i) do dec(i);

      // Now, if . ! ? or start of par uppercase it!
      if (pos_wstart>=0) and
        ((i<=0) or (par.CharItem[i]='.') or
          (par.CharItem[i]='!') or
          (par.CharItem[i]='?'))
        // Check for abbreviation (simplified - only check a..z
        and ((par.IsSpace(i-1) or (par.CharItem[i]<>'.') or
          ((par.CharItem[i-1]>='a') and (par.CharItem[i-1]<='z'))))
        then
        begin
          if Integer(par.CharItem[pos_wstart])<256 then
          begin
            s := AnsiUpperCase(Char(par.CharItem[pos_wstart]));
            if s<>' ' then par.CharItem[pos_wstart] := WideChar(s[1]);
          end
          else par.CharItem[pos_wstart] := WideUpperCase(par.CharItem[pos_wstart])[1];
        end;
      end;
    end;
  end;

```

new WPTools 7 Feature:

On popular demand the mode `wpAutoCapitalize` has been added to property [EditOptionsEx](#).

If this mode is active the editor will automatically capitalize Words at the beginning of a sentence.

7.8 J) Label Printing

WPTools 7 introduces an exciting new feature. This is the integrated label design and printing. When label printing is activated, each logical page in the editor will be displayed and printed on an individual label. The user can freely edit the label sheet and mover the text cursor from label to label. This makes it very easy to preview the labels which are about to be printed and make last changes.

The label printing is controlled by the interface [WPRichText1.RTFData.LabelDef](#).

Note: This feature is much more versatile than label printing using the demo [Print Labels \(TWPSuperPrint\)](#) since here the preview is editable.

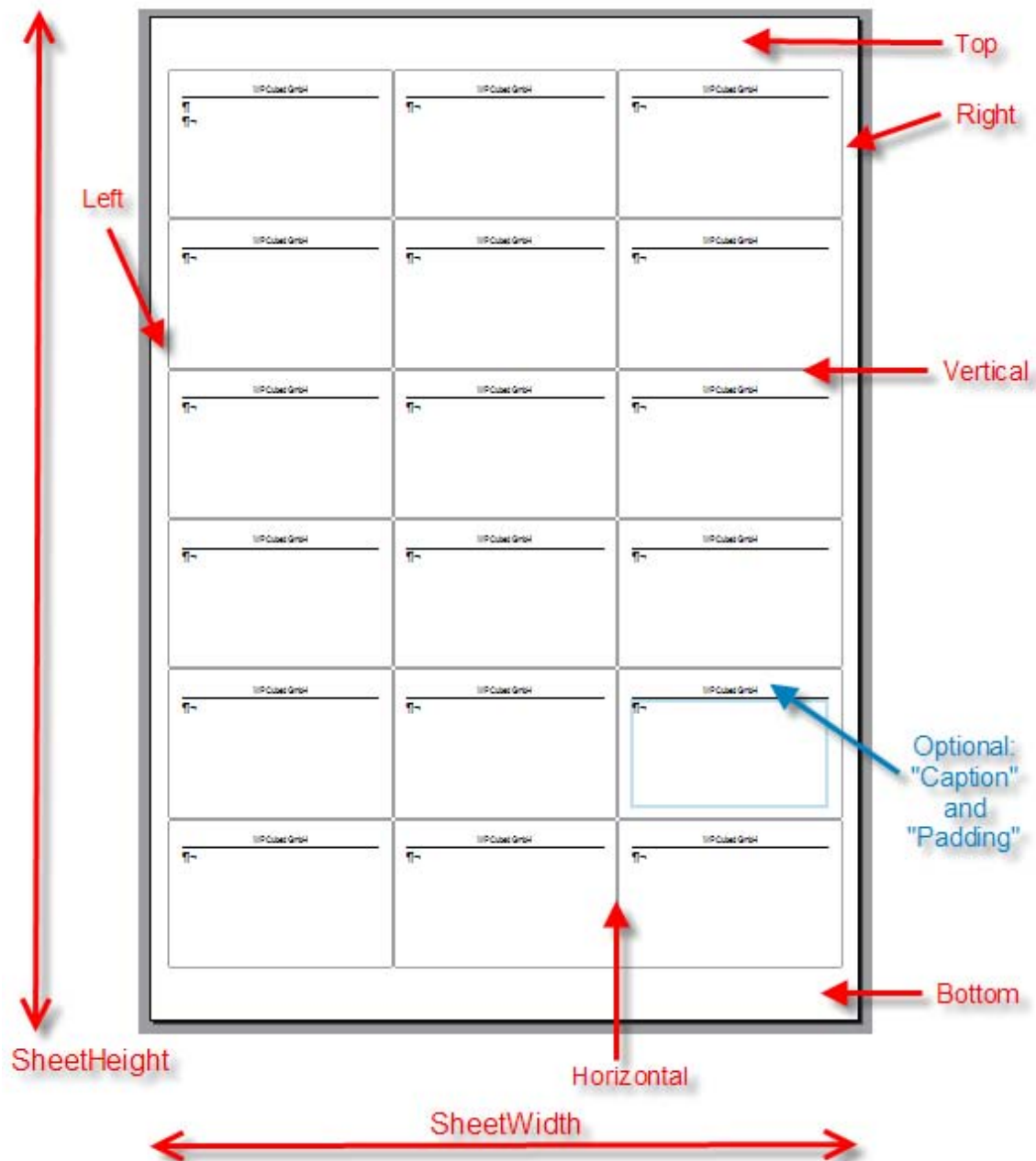
7.8.1 LabelDef

Using the new property LabelDef which was added to WPTools 7 You can quickly print labels. It is also possible to preview the label sheets just like they would be printed. It is even possible to edit the text on the label sheets. You can also specify the label number to start with. All parameters of a label can be specified, using centimeter or inch values, depending on [UnitIsInch](#).

A Label is either defined by the sheet size (width/height), column count, rowcount and the margins (top, bottom,left, right, horizontal and vertical)

or, alternatively

by the sheet size (width/height), the top, left, right and bottom margin and the specified label width and label height.



Overview:

[Active](#) : Switch Label display on / off

[AsText](#) : Retrieve and set the label definition as text

[UnitIsInch](#) : If true all floating point values are inch instead of centimeter (cm)

[Bottom](#) : Bottom Margin

[Caption](#) : Caption of the label, displayed over text

[ColumnCount](#) : Count of columns

[Horizontal](#) : Horizontal margin between columns

[LabelHeight](#) : Size of a label, if you set it you cannot change vertical margin and row count

[LabelWidth](#) : Size of a label, if you set it you cannot change horizontal margin and row count

[Left](#) : Left Margin

[Name](#) : Name of this label definition (encoded into "AsText")

[Padding](#) : Padding inside of the label

[Right](#) : Right Margin

[RowCount](#) : Count of rows

[SheetHeight](#) : Height of label sheet

[SheetWidth](#) : Width of label sheet

[StartNr](#) : Start nr for printing

[Top](#) : Top Margin

[Vertical](#) : Margin between rows

new: Caption and frame is not painted for empty labels.

New: When label printing is active, soft page breaks will be ignored.

You can switch on the old behavior with the format option

wpfLabelAllowSoftPagebreakLabelprinting:

Hint:

You can use the TWPValueEdit to build the userinterface.

This component implements the function `ValueAsInchOrCM()` to read the value as inch or cm.

So you can code as easy as

```
WPRichText1.RTFData.LabelDef.SheetWidth := SheetWidth.ValueAsInchOrCM(UseInch);
WPRichText1.RTFData.LabelDef.SheetHeight := SheetHeight.ValueAsInchOrCM(UseInch);
```

7.8.1.1 Active

Declaration

Active : Boolean;

Description

When this property is set to **true**, the editor will display a label sheet. The page size which is currently defined in the editor is overridden when this mode is on. The event `OnMeasurePage` is disabled.

7.8.1.2 UnitIsInch

Declaration

UnitIsInch : Boolean;

Description

If this value is true all floating point properties are interpreted as **inch** values, otherwise **CM** are expected.

7.8.1.3 SheetWidth

Declaration

SheetWidth : Single;

Description

The width of the label sheet (cm or Inch, depending on [UnitIsInch](#)).

7.8.1.4 SheetHeight

Declaration

SheetHeight : Single;

Description

The height of the label sheet (cm or Inch, depending on [UnitIsInch](#)).

7.8.1.5 Vertical

Declaration

Vertical : Single;

Description

The vertical margin between labels (in CM or inch).

7.8.1.6 Top

Declaration

Top : Single;

Description

The margin at the top of the label sheet.

7.8.1.7 StartNr

Declaration

StartNr : Integer;

Description

The number of the label on first page to start with. This property is useful if you work with half full label sheets.

7.8.1.8 RowCount

Declaration

RowCount : Integer;

Description

The count of labe rows.

7.8.1.9 Right

Declaration

Right : Single;

Description

The margin to the right of the label sheet.

7.8.1.10 Padding

Declaration

Padding : Single;

Description

The margin between text and label borders on all sides of the label. If this property is 0, the labels will be displayed in the preview as simple rectangles, if it is >0 a round rectangle will be drawn.

7.8.1.11 Name

Declaration

Name : String;

Description

The name of the label. Will be saved with property AsText.

7.8.1.12 Left

Declaration

Left : Single;

Description

The margin on the left of the label sheet.

7.8.1.13 LabelWidth

Declaration

LabelWidth : Single;

Description

The width of the label. You can read this property to display the current width.

If written, the properties ColumnCount and Horizontal are changed accordingly.

7.8.1.14 LabelHeight

Declaration

LabelHeight : Single;

Description

The height of the label. You can read this property to display the current height.

If written, the properties RowCount and Vertical are changed accordingly.

7.8.1.15 Horizontal

Declaration

Horizontal : Single;

Description

The horizontal margin between labels (in CM or inch).

7.8.1.16 ColumnCount

Declaration

ColumnCount : Integer;

Description

The count of label columns on the sheet.

7.8.1.17 Caption

Declaration

Caption : string;

Description

An optional caption for each label. This caption is printed on each label in a small font (Arial Narrow). You can use it to specify the return address.

7.8.1.18 Bottom

Declaration

Bottom : Single;

Description

The height of the bottom margin in cm or inch.

7.8.1.19 AsText

Declaration

AsText : string;

Description

Retrieves or sets all parameters except for Caption, Active and StartNr as string. This methods makes it easy to load and save predefined label definitions.

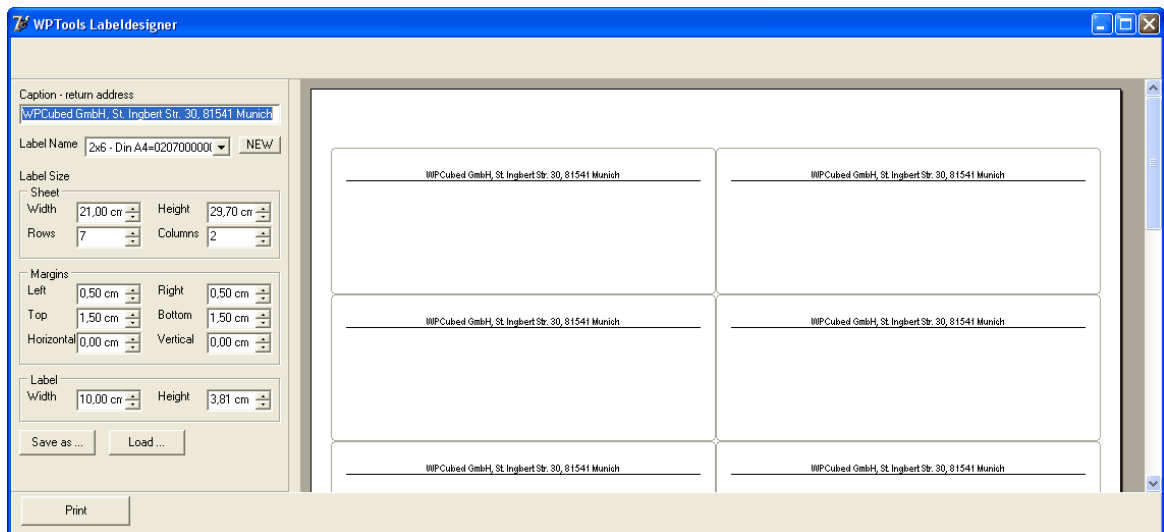
The format of the string is

Name=xx

with xx as the properties encoded into hexadecimal values. If required the unit (cm or inch) are converted as required.

7.8.2 Example Project

The demo label design contains the preview/edit windows and a TWPValueEdit control for each of the properties. It also allows saving and loading the definition and manages the list of definitions in the Items list of a TComboBox.



Please locate the Delphi project in directory Demos\J) LabelPrinting

7.8.2.1 Initialization

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    UseInch := false;
    // Width of the sheet
    WPRichText1.RTFData.LabelDef.SheetWidth := 21;
    WPRichText1.RTFData.LabelDef.SheetHeight := 29.7;
    // Rows and columns
    WPRichText1.RTFData.LabelDef.ColumnCount := 2;
    WPRichText1.RTFData.LabelDef.RowCount := 7;
    // Margins
    WPRichText1.RTFData.LabelDef.Left := 0.5;
    WPRichText1.RTFData.LabelDef.Right := 0.5;
    WPRichText1.RTFData.LabelDef.Top := 1.5;
    WPRichText1.RTFData.LabelDef.Bottom := 1.5;
    // Start with first
    WPRichText1.RTFData.LabelDef.StartNr := 0;
    // Return Address
    WPRichText1.RTFData.LabelDef.Caption := ReturnAdress.Text;
    // And activate
    WPRichText1.RTFData.LabelDef.Active := true;
    ReadProps;
end;
  
```

7.8.2.2 After changing sheet size and margins

```

procedure TForm1.MargHorizontalChange(Sender: TObject);
begin
    // UseInch ---- true: we work with inch, false: we work with CM
    WPRichText1.RTFData.LabelDef.UnitIsInch := UseInch;

    WPRichText1.RTFData.LabelDef.SheetWidth := SheetWidth.ValueAsInchOrCM(UseInch);
    WPRichText1.RTFData.LabelDef.SheetHeight := SheetHeight.ValueAsInchOrCM(UseInch);
  
```

```

WPRichText1.RTFData.LabelDef.Top := MargTop.ValueAsInchOrCM(UseInch);
WPRichText1.RTFData.LabelDef.Left := MargLeft.ValueAsInchOrCM(UseInch);
WPRichText1.RTFData.LabelDef.Right := MargRight.ValueAsInchOrCM(UseInch);
WPRichText1.RTFData.LabelDef.Bottom := MargBottom.ValueAsInchOrCM(UseInch);
WPRichText1.RTFData.LabelDef.Horizontal := MargHorizontal.ValueAsInchOrCM(UseInch);
WPRichText1.RTFData.LabelDef.Vertical := MargVertical.ValueAsInchOrCM(UseInch);

// Read
LabelWidth.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.LabelWidth, UseInch);
LabelHeight.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.LabelHeight, UseInch);

StoreProps;
WPRichText1.ReformatAll(true, true);
end;

```

7.8.2.3 Change of column count or row count

```

procedure TForm1.ColumnCountChange(Sender: TObject);
begin
    WPRichText1.RTFData.LabelDef.ColumnCount := ColumnCount.Value;
    WPRichText1.RTFData.LabelDef.RowCount := RowCount.Value;
    // Read
    LabelWidth.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.LabelWidth, UseInch);
    LabelHeight.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.LabelHeight, UseInch);

    StoreProps;
    WPRichText1.ReformatAll(true, true);
end;

```

7.8.2.4 Change of label width and height

Usually the labels are defined by the column count and row count property. But it is also possible to specify the width and height instead and the component will calculate the correct column and rowcount.

```

procedure TForm1.LabelWidthChange(Sender: TObject);
begin
    WPRichText1.RTFData.LabelDef.LabelWidth := LabelWidth.ValueAsInchOrCM(UseInch);
    WPRichText1.RTFData.LabelDef.LabelHeight := LabelHeight.ValueAsInchOrCM(UseInch);

    ColumnCount.SetValue( WPRichText1.RTFData.LabelDef.ColumnCount );
    RowCount.SetValue ( WPRichText1.RTFData.LabelDef.RowCount );

    StoreProps;
    WPRichText1.ReformatAll(true, true);
end;

```

7.8.2.5 ReadProps and StoreProps

ReadProps updates the visible controls

```

procedure TForm1.ReadProps;
begin
    SheetWidth.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.SheetWidth , UseInch);
    SheetHeight.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.SheetHeight , UseInch);
    LabelWidth.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.LabelWidth , UseInch);
    LabelHeight.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.LabelHeight , UseInch);

    MargTop.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.Top , UseInch);
    MargLeft.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.Left , UseInch);
    MargRight.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.Right , UseInch);
    MargBottom.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.Bottom , UseInch);
    MargHorizontal.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.Horizontal , UseInch);

```

```

    MargVertical.SetValueAsInchOrCM( WPRichText1.RTFData.LabelDef.Vertical , UseInch);

    ColumnCount.Value := WPRichText1.RTFData.LabelDef.ColumnCount;
    RowCount.Value := WPRichText1.RTFData.LabelDef.RowCount;

    LabelName.Text := WPRichText1.RTFData.LabelDef.Name;
end;

```

StoreProps saves the definition to the correct item in a string list. "Name" is used to identify the item.

```

procedure TForm1.StoreProps;
var i : Integer;
begin
    i := LabelName.Items.IndexOfName(WPRichText1.RTFData.LabelDef.Name);
    if i>=0 then
        LabelName.Items[i] := WPRichText1.RTFData.LabelDef.AsText
    else
        begin
            i := LabelName.Items.Count;
            LabelName.Items.Append(WPRichText1.RTFData.LabelDef.AsText);
        end;
        LabelName.ItemIndex := i;
end;

```

7.9 H) Techniques

7.9.1 Styles

7.9.1.1 Shared Styles and Style Scroller

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.

Please note that borders can not be defined in styles.

Text styles usually have names, for example "H1", "H2" or "Body". 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.

The default attribute will be used for all the text which does not have an attribute in its own or is not using a style.

In this demo we initialize the default attribute using:

```

WPRichText1.DefaultAttr.SetFontName('Arial');
WPRichText1.DefaultAttr.SetFontSize(11);
WPRichText1.WritingAttr.Clear;

WPRichText2.DefaultAttr.SetFontName('Courier New');
WPRichText2.DefaultAttr.SetFontSize(11);
WPRichText2.WritingAttr.Clear;

```

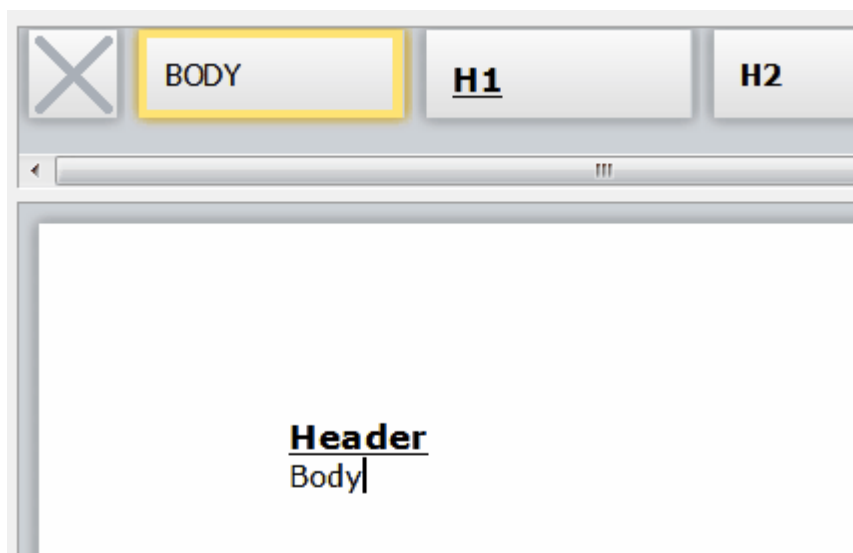
It is very easy to share the same paragraph style sheet in several TWPRichText editors.

Simply drop a TWPRTFProps component on the form and add a reference to this instance in the property WPRTFPropsComponent of each of the TWPRichText.

You can double click on the **WPRTFPropsComponent** to create an initial set of styles.

Please note that a "WPRichText.Clear" will clear the styles, too. You need to use ClearEx to only delete the text.

The TWPStyleScroller inherits also from the editor class, but displays the styles as read-only tiles. If property "HasDefaultItem = true" it will display a first square item with an X. That is used to set the style reference to unassigned. Using property "Shaded" the design of the scroller can be switched. The property Aspect tells the component the aspect of the tiles, $H=Aspect*W$.



You need to set the property WPRTFPropsComponent of the WPStyleScroller, too. To link it to the editor(s) create a TActionList and add a WPToolsCustomEditContolAction. As property AttachedControl specify WPStyleScroller1. All affected editors should list the ActionList in property ActionList. This makes a 2 way update possible - the style scroller will highlight the current style and a click will change the style.

Hint:

In case you do not need different editors to share the same styles, you do not need an instance of the TWPRTFProps component. TWPRichText will create its own.

In this case, just add one line the the Form.OnCreate event to make the StyleScroller work:

```
WPStyleScroller1.WPRTFPropsComponent := WPRichText1.RTFData.RTFProps;
```

With shared RTFProps it is easy to copy the current paragraph to a different TWPRichText

```
var par : TParagraph;

par := WPRichText1.ActiveParagraph.CreateCopyList(true, WPRichText1.ActiveParagraph);
WPRichText2.ActiveParagraph.NextPar := par;
WPRichText2.ReformatAll(false, true);
```

You can also copy the current table row to a table in a different TWPRichText

```
var par, aTable : TParagraph;

// Copy this row
par := WPRichText1.TableRow;
par := par.CreateCopyList(true, par);

// Insert a row - if necessary create a surrounding table object
// Insert after current row
if WPRichText2.TableRow<>nil then
    WPRichText2.TableRow.NextPar := par
else
begin
    // Create a new table
    aTable := WPRichText2.ActiveParagraph.AppendNewTable(false);
    // and insert the row
    aTable.AppendChild(par);
end;
```

7.9.1.2 Read/Write CSS

With WPTools 7 the TWPTextStyle object, which is also the ancestor of TParagraph (see [Datastructures](#)) has the methods AGet_CSS and ASet_CSS to read and write a CSS compatible style definition. The name and parentheses are not used in this case.

Also the style collection has an easy way to load and save cascading style sheets, GetCSS and SetCSS. All those methods are implemented in unit WPIOCSS which has to be linked in unless it isn't already by unit WPIO.

Please note, when you changed styles by directly accessing the TWPTextStyle or the style collection you need to call **ReformatAll(true, true)** to format and display the text. Since a TWPStyleScroller is also not informed about the change, you need to call UpdateStyleList.

Example:

```
procedure TForm1.GetSingleStyleCSS(Sender: TObject);
begin
    if WPStyleScroller1.CurrentStyle<>nil then
        CSSMemo.Text := WPStyleScroller1.CurrentStyle.AGet_CSS(true, false,
end;
```

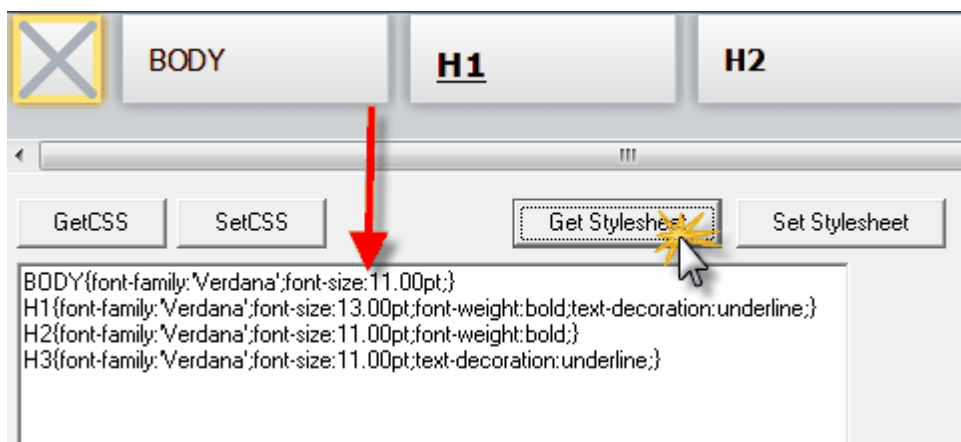
```

procedure TForm1.SetSingleStyle(Sender: TObject);
begin
    if WPStyleScroller1.CurrentStyle<>nil then
        begin
            WPStyleScroller1.CurrentStyle.ASet_CSS(CSSMemo.Text);
            // We need to update format and screen
            WPRichText1.ReformatAll(true,true);
            WPRichText2.ReformatAll(true,true);
            WPStyleScroller1.ReformatAll(true,true);
        end;
    end;

procedure TForm1.GetStylesheetCSS(Sender: TObject);
begin
    CSSMemo.Text := WPRTFProps1.ParStyles.GetCSS;
end;

procedure TForm1.SetStylesheetCSS(Sender: TObject);
begin
    WPRTFProps1.ParStyles.SetCSS(CSSMemo.Text);
    // We need to update format and screen
    WPRichText1.ReformatAll(true,true);
    WPRichText2.ReformatAll(true,true);
    WPStyleScroller1.UpdateStyleList;
end;

```



The TWPRichText implements some methods to load and save CSS. (Internally the methods of ParStyles are called.)

This method saves paragraph styles to a CSS style sheet.

function **SaveCSSheet**: string;

This method saves paragraph styles to a CSS style sheet file.

procedure **SaveCSSFile**(CSSFileName: string);

The following TWPRichText methods also call ReformatAll:

This method loads paragraph styles from a CSS style sheet file.

```
function LoadCSSFile(CSSFileName: string; Merge : Boolean = false): Integer;
```

This method loads paragraph styles from a CSS style sheet.

```
function LoadCSSheet(StyleString: string; Merge : Boolean = false): Integer;
```

Important:

WPTools also supports the proprietary **WPCSS** - this format is similar to CSS but optimized for performance. It can hold all text features WPTools supports. You can get and set WPCSS in styles, paragraphs, attribute interfaces etc. It is also use the WPT format to define the attributes.

It is incompatible to the standard CSS though and can only be used in WPTools or TextDynamic.

Example WPCSS description of a style sheet,
which was read by CSSMemo.Text := WPRTFProps1.ParStyles.GetWPCSS:

```
BODY=CharFont:'Verdana';CharFontSize:1100;  
H1=CharFont:'Verdana';CharFontSize:1300;CharStyleMask:5;CharStyleON:5;  
H2=CharFont:'Verdana';CharFontSize:1100;CharStyleMask:1;CharStyleON:1;  
H3=CharFont:'Verdana';CharFontSize:1100;CharStyleMask:4;CharStyleON:4;
```

The names represent the WPAT_... property IDs used for the respective properties. (Please see [reference](#) for supported WPAT_ codes.)

To read a WPCSS string of a style 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.

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);
```

7.9.1.3 Add and Assign style

In this demo we create a style (sty : TWPTTextStyle;) by adding it to the style collection.

```
sty := WPRTFProps1.ParStyles.AddStyle(NewStyleName.Text);
```

In case you do not use a TRTFProps component simply use this

```
sty := WPRichText1.ParStyles.AddStyle(NewStyleName.Text);
```

Both does the same and is accessing the same ParStyles collection.

Then You can assign properties using the ASet methods:

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

To make the style scroller "see" the new style you need to add a

```
WPStyleScroller1.UpdateStyleList;
```

Please see [reference](#) for supported WPAT_ codes.

Many developers 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 := NewStyleName.Text;`

or the style directly:
`par.ABaseStyle := sty;`

par is a reference to a TParagraph object. That can be WPRichText1.ActiveParagraph or WPRichText.FirstPar;

WPTools supports the definition of various paragraph attributes in styles, but no tabstops.

When you assign a style to a paragraph, you often cannot see the effect right away.

The reason for this is, that the paragraph itself and its text already defines certain attributes and so overrides the attributes which are defined in the style.

a) To remove the attributes which are defined by a style you can use the method **SetStyle** which can be used with a style number, a style name or the reference to an item of the ParStyles collection.

Two optional parameters control if the paragraph attributes and/or the character attributes should be adjusted to make all style attributes visible.

`par.SetStyle(stylename , true, true);`

b) If you are working with selected text, you can use the procedure **SelectedTextAttr.ClearAttr**(false,true) to remove all styles from the selected text, only the paragraph styles will be used.

You can also use **SelectedTextAttr.ClearAttrOverride** - this will remove the attributes which are used in the paragraph styles. So the attributes which are not override attributes will be preserved, the attributes which will otherwise hide the paragraph style attributes will be removed.

c) You can also call `par.ClearCharAttributes(true)` to remove the character attributes from all characters.

After assigning a style the text should be reformatted. WPTools 7 will take care that only the modified paragraphs are also initialized.

Example:

```
WPRichText1.ActiveParagraph.ABaseStyleName := NewStyleName.Text;
WPRichText1.ActiveParagraph.ClearCharAttributes;
WPRichText1.ReformatAll(false, true);
```

This code will simply attach a style, it will not modify any properties of the paragraph or the text in this paragraph. So it is possible that the text does not use certain attributes of the style since they are overridden.

7.9.1.4 Work with SPAN Styles

Especially to provide strong HTML support WPTools can work with embedded SPAN styles. These 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 in 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).

7.9.1.5 Numbering

In WPTools Version 7 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;
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;

```

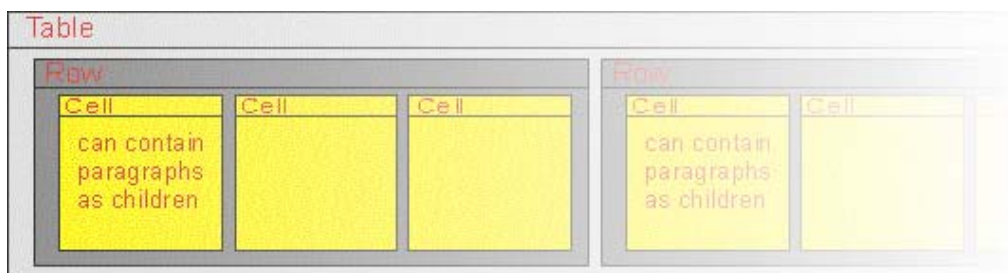
Note about CurrAttr: This is an interface which was introduced in WPTools 3. It is still supported and implemented in unit WPCTRRich. If you do not want to link unit WPCTRRich you cannot use CurrAttr. For new code we suggest to use ActiveParagraph, ASet or SelectTextAttr methods.

7.9.2 Tables

WPTools Version 7 tables are very powerful:

1	2	3	4	5	6	7	8	9	10	Overall suitability index Based on aptitude tests and personality questionnaire
Low overall reliability	Below average	X	Slightly below average	Average	Slightly above average	Well above average	Outstanding			

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;
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.

7.9.2.1 Set width in inch of a certain column

Note: The function TParagraph._IsWidth_tw returns the current width of a cell in twips. Please note that this function requires the text to be formatted.

Example 1)

In this example we do not only set the width of a certain column but adjust the width of the parent table to fit in all columns:

```

var cellwidth_tw, newwidth_tw : Integer; // Twip Values!
    par, cell, row : TParagraph;
begin
  // cellwidth_tw := WPValueEdit1.Value; // Use a WPValueEdit to enter the
  // number
  cellwidth_tw := WPInchToTwips(2.5) // use a fixed INCH value

  row := WPRichText1.TableRow;
  cell := WPRichText1.ActiveParagraph.Cell;
  if (row=nil) or (cell=nil) then
    ShowMessage('Please position the cursor in a cell!')
  else

```

```

begin
    // Calculate current width of all columns
    newwidth_tw := 0;
    par := row.ChildPar; // this are the cells in this row
    while par<>nil do
        begin
            if par=cell then inc(newwidth_tw, cellwidth_tw )
            else inc( newwidth_tw, par._IsWidthTw );
            par := par.NextPar; // Sibling cell !
        end;
        // Make sure we do not use % values !
        row.ParentTable.FixAllCellWidths(0);
        // Assign TWIPS value to THIS column
        cell.ASetColumn(WPAT_COLWIDTH, cellwidth_tw);
        // And set the table width to fit all coluns
        row.ParentTable.ASet(WPAT_BoxWidth, newwidth_tw);
        // Reformat
        WPRichText1.DelayedReformat;
    end;
end;

```

Example 2)

This extended example creates an array of TWPValueEdit controls in a scrollbox to let the user see and adjust the width of each individual column:



```

// This is the UpdateGrid procedure
procedure TForm1.UpdateWidthGridClick(Sender: TObject);
var i: Integer;
    cell, row: TParagraph;
    ctrl: TWPValueEdit;
begin
    // Empty the scrollbox
    for i := ScrollBox1.ControlCount - 1 downto 0 do
        with ScrollBox1.Controls[i] do
            begin
                Parent := nil;
                Free;
            end;

    // Fill the scrollbox
    row := WPRichText1.TableRow;
    cell := WPRichText1.ActiveParagraph.Cell;
    if (row = nil) or (cell = nil) then
        ShowMessage('Please position the cursor in a cell!')
    else
        begin
            // Make sure we do not use % values !
            row.ParentTable.FixAllCellWidths(0);
            // Create ColCount edits:
            for i := 0 to row.ColCount - 1 do
                begin
                    ctrl := TWPValueEdit.Create(ScrollBox1);
                    ctrl.Left := i * 80;
                    ctrl.Width := 78;
                end;
        end;
    end;

```

```

        ctrl.Parent := ScrollBox1;
        ctrl.Tag := i; // ColNr
        ctrl.UnitType := euInch; // euCm for Centimeters
        ctrl.Value := row.Cols[i]._IsWidthTw;
        ctrl.OnChange := CellValueEditChange;
    end;
end;
end;

// OnChange event handler for each TWPValueEdit in array
procedure TForm1.CellValueEditChange(Sender: TObject);
var i, w: Integer;
    cell: TParagraph;
begin
    if WPRichText1.TableRow <> nil then
        cell := WPRichText1.TableRow.Cols[(Sender as TWPValueEdit).Tag]
    else cell := nil;

    if cell <> nil then
    begin
        // Calculate sum of all width
        w := 0;
        for i := 0 to ScrollBox1.ControlCount - 1 do
            inc(w, TWPValueEdit(ScrollBox1.Controls[i]).Value);
        // and apply to table

        cell.ParentTable.ASet(WPAT_BoxWidth, w);
        // and apply to column
        cell.ASetColumn(WPAT_COLWIDTH, (Sender as TWPValueEdit).Value);
        // Reformat
        WPRichText1.DelayedReformat;
    end;
end;

```

7.9.2.2 Work with % width

WPTools Version 7 also has the ability to use % values for column widths. The procedure `WPRichText1.TableMakeCellWidthPC` converts all tables in the document to measure the column width in %.

If the flag `wpAlwaysColWidthPC` was set in property `EditOptionsEx` this procedure is executed after a file was loaded and after each change to the column width using the mouse.

The % column width is stored as property `WPAT_ColWidthPC` as `%*100`.

```

// Set the width in 5, example 10%
WPRichText1.ActivePar.ASetColumn(WPAT_COLWIDTH_PC, 1000);
// Delete the fixed width since it has priority
WPRichText1.ActivePar.ADelColumn(WPAT_ColWidth);
// Reformat the text
WPRichText1.DelayedReformat;

```

7.9.2.3 Useful code samples for table handling

This example creates a new row when ENTER is pressed at the last position of a row.

```

procedure TWPEditor.WPRichText1KeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
begin

```

```

if (Key=VK_RETURN) and
  (WPRichText1.ActivePosInPar>=WPRichText1.ActiveParagraph.CharCount)and
  (WPRichText1.TableCell<>nil) and (WPRichText1.TableCell.NextPar=nil) then
begin
  WPRichText1.TableRow.Duplicate(false, true, false);
  WPRichText1.ActiveParagraph := WPRichText1.TableRow.NextPar.ColFirst;
  // Duplicate is low level - we need to call Reformat manually
  WPRichText1.ReformatAll(false, true);
  Key := 0;
end;
end;

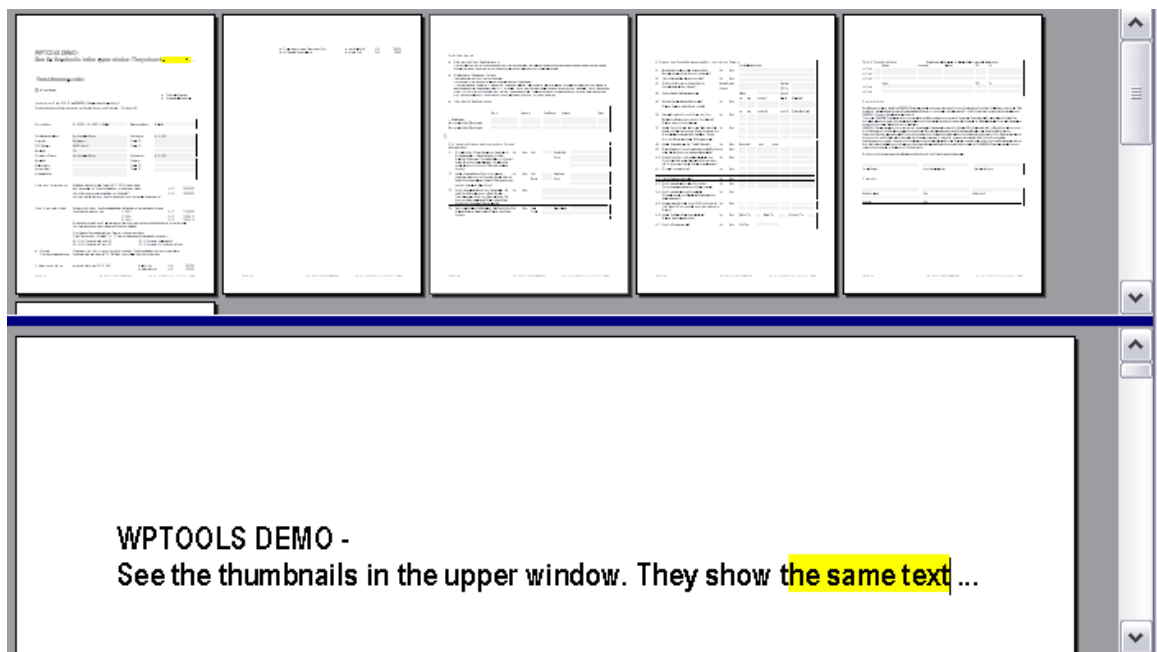
```

7.9.3 Multiple Editors for the Same Text

WPTools Version 7 allows the use of multiple editors which all work with the same text.

This is also known as the split screen technique.

Example:



In case you "only" need to share Styles, You can use the TWPRTFProps component as highlighted in the [Styles demo](#).

7.9.3.1 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.

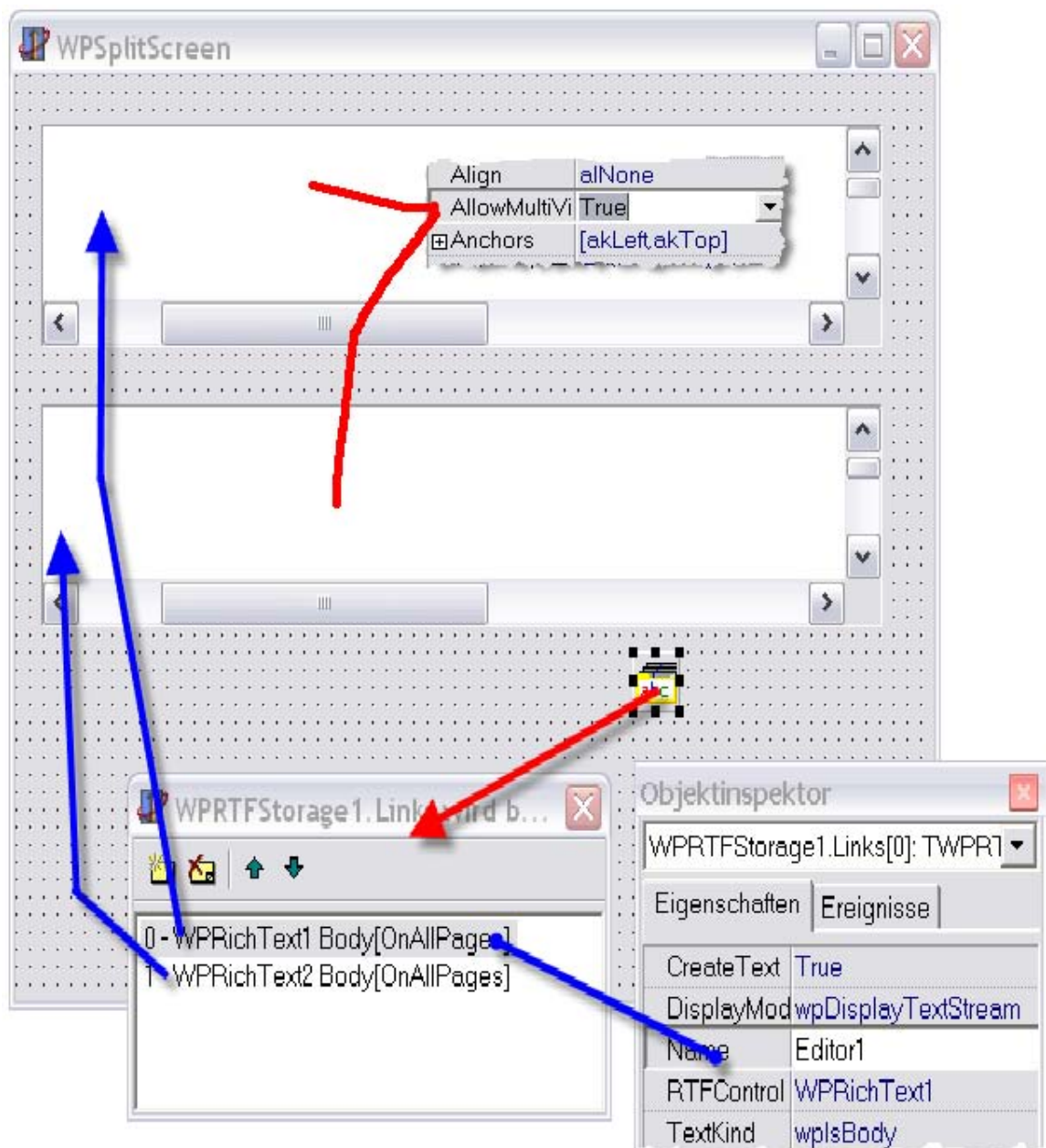
```
procedure TForm1.OnInitRTFData(Sender: TObject;  
    var RTFDataObject: TWPRTFDataCollection;  
    var RTFPropsObject: TWPRTFProps);  
begin  
    if RTFData = nil then  
        begin  
            RTFData := TWPRTFDataCollection.Create(TWPRTFDataBlock);  
            RTFDataProps := TWPRTFProps.Create;  
            RTFData.RTFProps := RTFDataProps;  
        end;  
        RTFDataObject := RTFData;  
end;
```

7.9.3.2 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.



7.9.4 Simulated MDI (one editor, multiple documents)

If you need to let the user edit and switch between documents, You do not need to create multiple TWPRichText.

You only need one TWPRichText and switch the attached RTFDataCollection.

We enhanced the "[mini](#)" editor to do so. We just added a tabset to switch between the documents, some code to intercept the OPEN, CLOSE and NEW actions of the TWPToolbar and of course a list with RTFData objects handled by TStringlist.

Variables:

```

type
  TForm1 = class(TForm)
  ...
  public
    FRTFDataCollections : TStringList;
    function AddRTFData(name : string) : Integer;
    procedure InitTabset;
    procedure DelRTFData;
  end;

var
  Form1: TForm1;

```

implementation

```
{ $R *.DFM }
```

Add a document

```

function TForm1.AddRTFData(name : string) : Integer;
var element : TWPRTFDataCollection;
begin
  element := TWPRTFDataCollection.Create(TWPRTFDataBlock);
  element.MakeRTFProps;
  Result := FRTFDataCollections.Count;
  FRTFDataCollections.AddObject(name, element);
  WPRichText1.RemoveRTFData;
  WPRichText1.SetRTFData(element);
  WPRichText1.ReformatAll(false, true);
  InitTabSet;
end;

```

Delete current document, clear if it is last

```

procedure TForm1.DelRTFData;
var i      : Integer;
    element : TWPRTFDataCollection;
    bNeedClear : Boolean;
begin
  bNeedClear := true;
  if FRTFDataCollections.Count>1 then
    begin
      i := FRTFDataCollections.IndexOfObject( WPRichText1.RTFData );
      if i>=0 then
        begin
          element := WPRichText1.RTFData;
          WPRichText1.RemoveRTFData;
          FRTFDataCollections.Delete(i);
          if i>0 then dec(i);
          element.Free;
          element := TWPRTFDataCollection(FRTFDataCollections.Objects[i]);
          WPRichText1.SetRTFData(element);
          WPRichText1.ReformatAll(false, true);
        end;
    end;

```

```

        InitTabSet;
        bNeedClear := false;
    end;
end;
if bNeedClear then
begin
    WPRichText1.Clear;
    WPRichText1.CheckHasBody;
    WPRichText1.SetFocus;
end;
end;

```

Update the tabset "TabControl1" and display current filenames. Select the current tab.

```

procedure TForm1.InitTabset;
var i : Integer;
begin
    TabControl1.Tabs.Clear;
    for i := 0 to FRTFDataCollections.Count-1 do
        TabControl1.Tabs.Add(
            TWPRTFDataCollection(FRTFDataCollections.Objects[i]).LastFileName
        );
    i := FRTFDataCollections.IndexOfObject( WPRichText1.RTFData );
    if i>=0 then TabControl1.TabIndex := i
    else TabControl1.Tabs.Add('#');
end;

```

Switch between documents - tabset OnChange event.

```

procedure TForm1.TabControl1Change(Sender: TObject);
begin
    WPRichText1.SetRTFData(TWPRTFDataCollection(FRTFDataCollections.Objects[
end;

```

Update display of filename after SaveAs

This code is called by the TWPRichText after a file was saved. It makes it easy to update the GUI with the new filename. Since our method InitTabset is reading the filenames from the DataCollection objects, the handling is very easy.

```

procedure TForm1.WPRichText1ChangeLastFileName(Sender: TObject);
begin
    InitTabset;
end;

```

Intercept a click on toolbar button

This code is called by the TWPToolbar in the OnIconSelection event.

```

procedure TForm1.WPToolbar1IconSelection(Sender: TObject;
    var Typ: TWpSelNr; const str: String; const group, num, index: Integer);
begin
    if (typ=wptIconSel) and (group=WPI_GR_DISK) then
        case num of

```

```

WPI_CO_Exit : Close;
WPI_CO_New  :
begin
    AddRTFData( '' );
    Typ := wptIconDeSel;
end;
WPI_CO_Open :
begin
    AddRTFData( '' );
    if not WPRichText1.Load then DelRTFData;
    Typ := wptIconDeSel;
end;
WPI_CO_Close:
begin
    if WPRichText1.CanClose then
        DelRTFData;
        Typ := wptIconDeSel;
    end;
end;
end;

```

Clean up the documents

```

procedure TForm1.FormDestroy(Sender: TObject);
var i : Integer;
begin
    WPPreview1.WPRichText := nil;
    WPRichText1.RemoveRTFData;
    for i:=0 to FRTFDataCollections.Count-1 do
        FRTFDataCollections.Objects[i].Free;
    FRTFDataCollections.Free;
end;

```

Initialize the document list in Form.OnCreate

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    FRTFDataCollections := TStringList.Create;
    AddRTFData('');
    ...

```

7.9.5 Watermarks

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

In WPTools Version 7 Watermarks have to be printed inside the event handler for **OnPaintWatermark**.

This event gets this parameters:

Sender	a reference to the TWPRichText component which triggered 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 printtime 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 <code>RTFEngine.DisplayedText.Pages[RTFPageNr - 1]</code> .
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 <code>PageRect.Left</code> or <code>PageRect.Top</code> .

```

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 <code>'wppInPaintDesktop'</code> you can write code which is not active at print time.

Example:

You can print the contents of one editor to the background of a different editor: (demo WaterM3)

```

procedure TWPLetterHeadEdit.WPRichText1PaintWatermark(Sender: TObject;
    RTFEngine: TWPRTFEnginePaint; toCanvas: TCanvas; PageRect: TRect;
    PaintPageNr, RTFPageNr: Integer; WaterMarkRef: TObject; XRes,
    YRes: Integer; CurrentZoom: Single; PaintMode: TWPPaintModes);
begin
    // We are painting on a RTF-Engine surface so use the
    // PaintPageMode wpNoViewPortAPI or wpUseWorldScaling since everything has been
    // set up already
    WPLetterhead.PaintPageOnCanvas(
        0, // PageNo
        0, 0, 0, 0, // Use current page size
        toCanvas, // The destination canvas
        PaintMode, // or []
        XRes, YRes, // the printing resolution
        -1, -1, // No clipping
        [wpUseWorldScaling]);
end;

```

Other examples how Watermarks can be used (Project: Demos\Tasks\Watermark)

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;
    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;
end;

```

7.9.6 PaintEngine

7.9.6.1 Superprint: Print Booklets and Labels

We've added a new component, `TWPSuperPrint`, that allows you to centrally manage a variety of 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

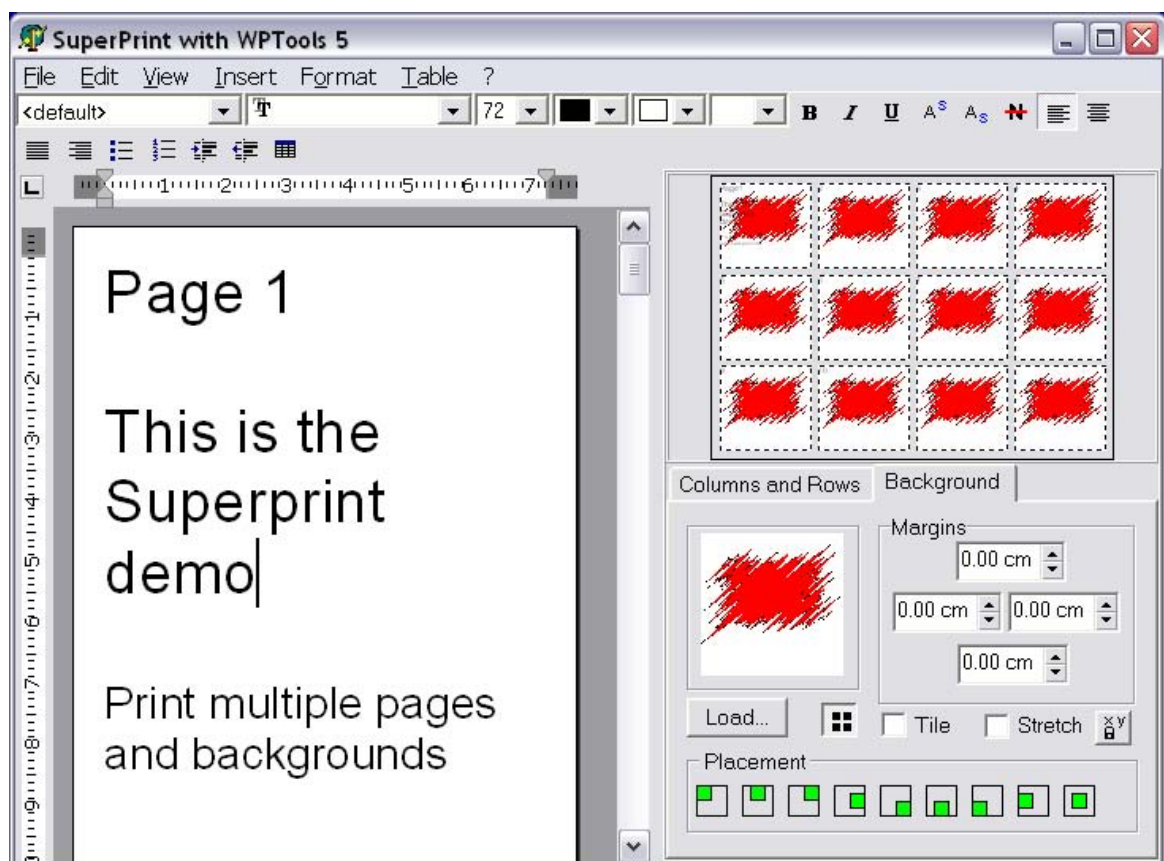
"two up" (two pages per sheet) booklet printing

- 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:

Sheet #	Page # on	
	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

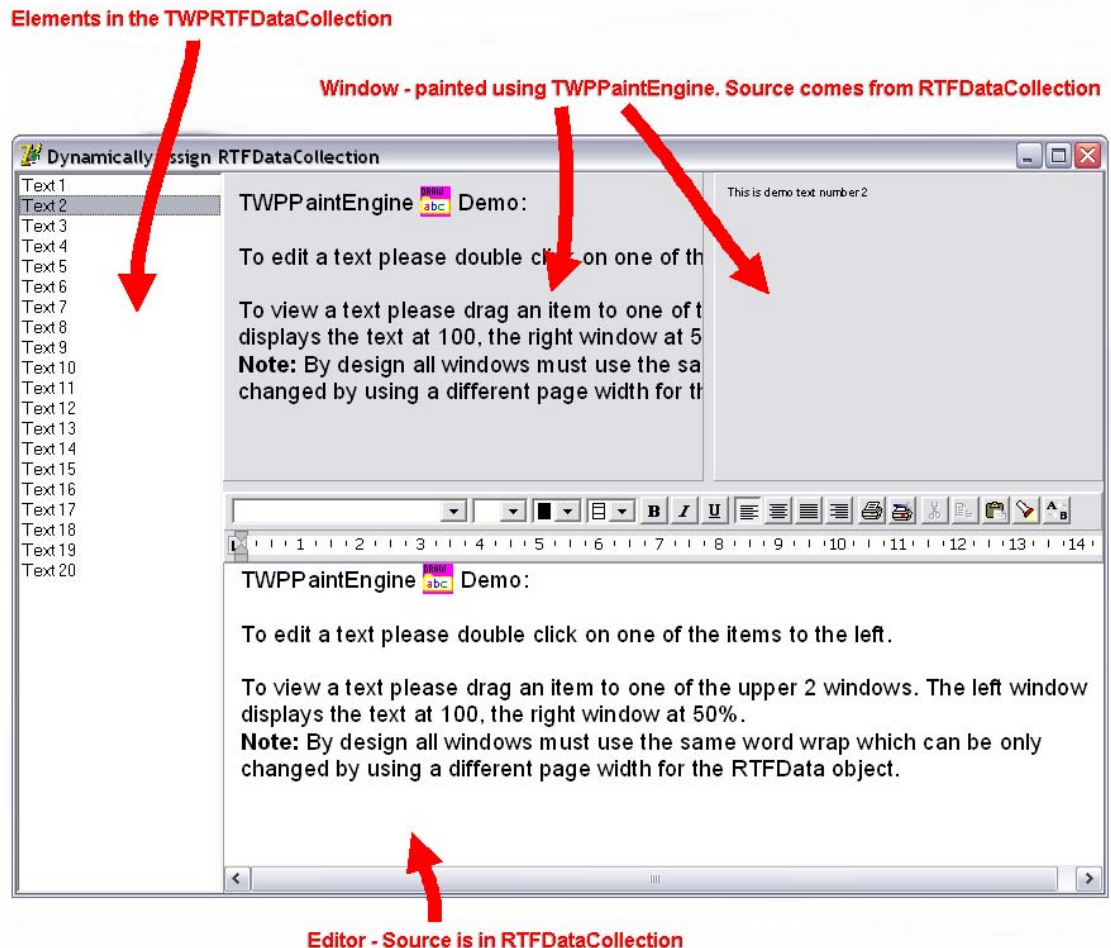
To print labels, first, create the label (from within the TWPRichText component you assigned to the WPSuperPrint1.EditBox property), or import one into there. Then (in no particular order)

- 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!

7.9.6.2 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!



This code is used to assign the text which should be edited:

```

procedure TWPDynRTFData.DynDataListDbClick(Sender: TObject);
begin
    // Remove link to this RTF Data Object
    WPRichText1.RemoveRTFData;
    // And add link to this one
    WPRichText1.SetRTFData(
        DynDataList.Items.Objects[DynDataList.ItemIndex]
        as TWPRTFDataCollection);
    // Make sure the new text is shown
    WPRichText1.SetFocus;
end;

```

The boxes are painted using **TWPPaintEngine** components which have been created in code in "FormCreate".

To change the RTFData which they are using this code is used after Drag&Drop:

```

paint1.RTFData :=

```

```
DynDataList.Items.Objects[DynDataList.ItemIndex]  
as TWPRTFDataCollection;
```

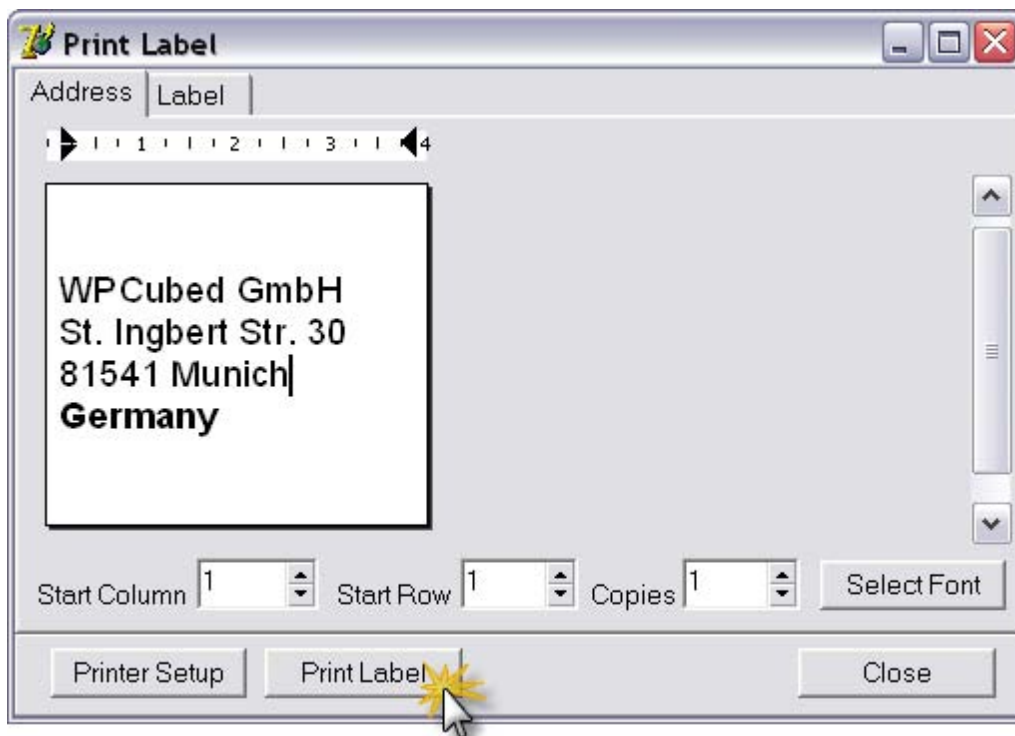
7.9.6.3 Print Labels (TWPSuperPrint)

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

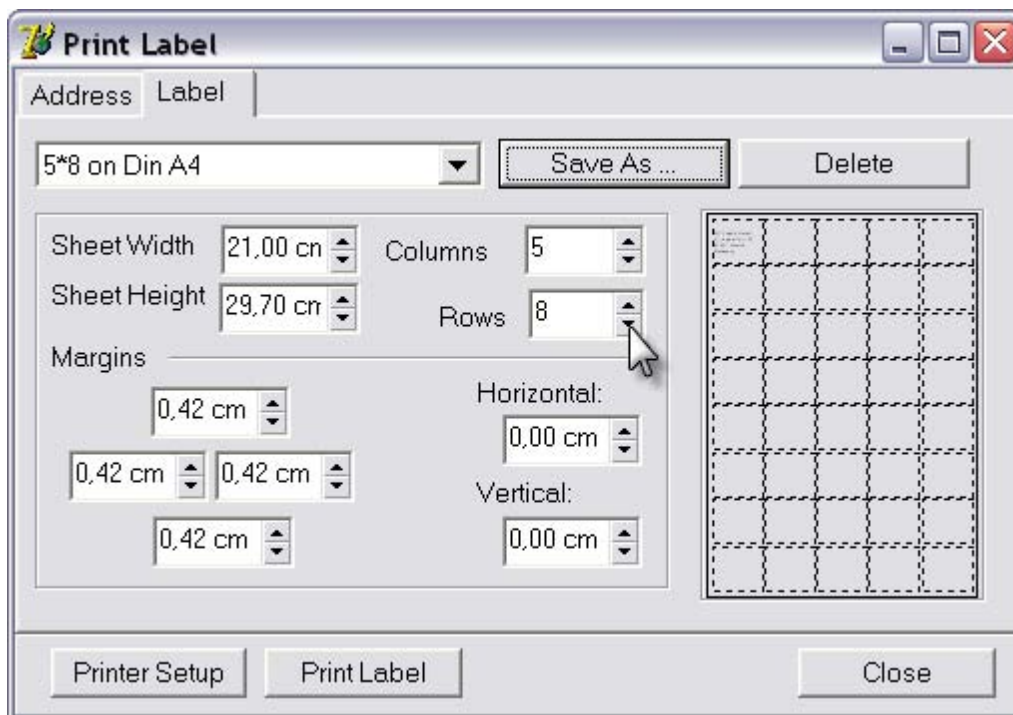
Note: Please also see the [new integrated label printing](#) feature which was added to WPTools 7!

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>
</Labels>
```

Note: The horizontal and vertical Margins are the small gaps BETWEEN labels - they are not the "pitch".

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, // Multiplier 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 accurate. Of course these offsets can be used to adjust the printing.

7.9.7 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;
```

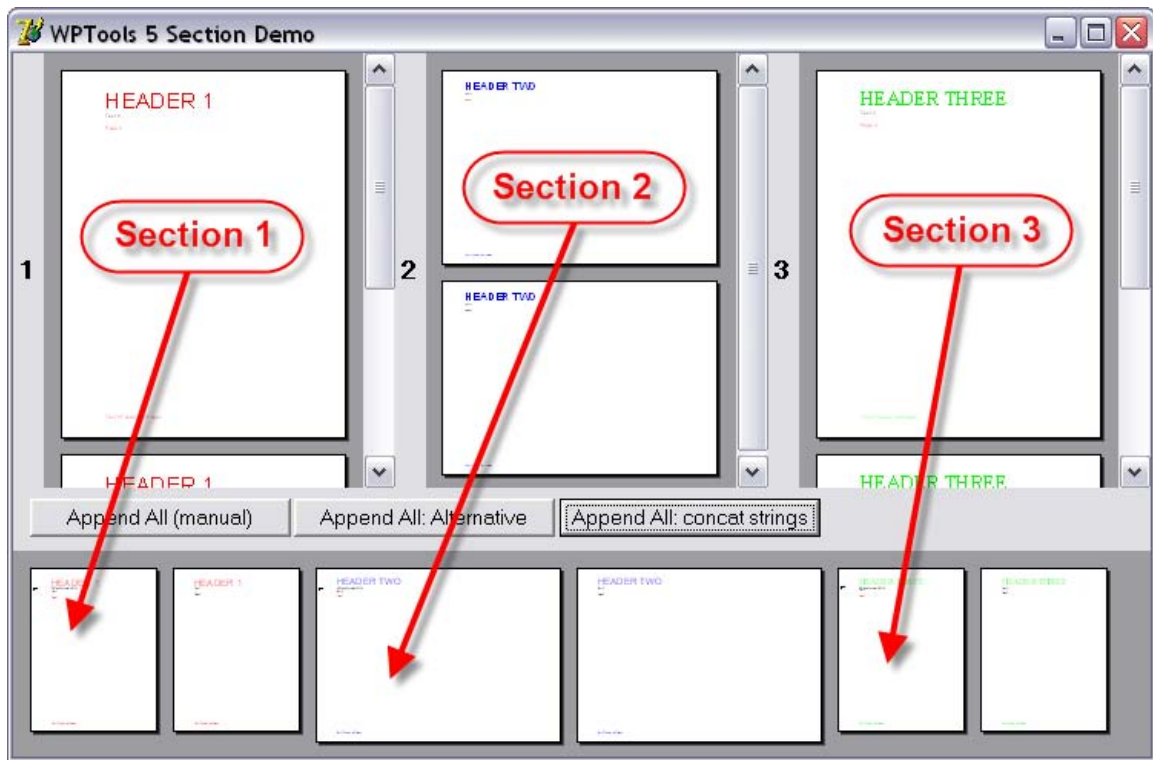
The following flags exist in property Select:

```
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 Version 7 does not work well with different page sizes yet. So please switch this feature off using WPAII.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 3 different approaches:

- [1.\) Full implementation - shows how to work with sections](#)
- [2.\) Use utility procedure AppendAsSection - same functionality as 1\)](#)
- [3.\) Use strings in WPTOOLS format with the <newsection/> tag.](#)

4) Append section with individual footer (or header)

7.9.7.1 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.AsANSISString('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->CPosition = MaxInt;
  this->WPRichText1->SelectionAsString = Source->AsANSIString("WPTOOLS");
}

```

7.9.7.2 Use utility procedure AppendAsSection

```

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

```

7.9.7.3 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;

```

7.9.7.4 Append section with individual footer (or header)

You can use WPRichText1.ActiveParagraph.StartNewSection to add a new section property object to the current paragraph. The return object will be an object of class TWPRTFSectionProps. In property 'Select' you can change which properties should be used for this section.

Using WPRichText1.HeaderFooter.Append you can append a new RTFDataBlock to be used as footer. 'Get' cannot be used, since this will reuse an existing RTFDataBlock.

```

procedure TForm1.PortraitLandscapeClick(Sender: TObject);
var sect : TWPRTFSectionProps;
    footer : TWPRTFDataBlock;
begin
  WPRichText1.CPosition := MaxInt;
  Inc(c);
  // start a section
  sect := WPRichText1.ActiveParagraph.StartNewSection;
  sect.Select := [wpsec_PageSize]; // 1. (in this order!)
  sect.Landscape := Sender = Landscape; // 2.

  // Now new page and some text
  WPRichText1.InputString(#12+#32+IntToStr(c)+#13);

```

```

// also add special header+footer for this section
footer := WPRichText1.HeaderFooter.Append(wpIsFooter,wpraOnAllPages, ''
);
footer.UsedForSectionID := sect.SectionID;

WPRichText1.ActiveText := footer;
WPRichText1.InputString('Section ' + IntToStr(c) + #9);
WPRichText1.InputTextFieldName('PAGE');

WPRichText1.ActiveParagraph.TabstopAdd(
    sect.PageWidth-sect.LeftMargin-sect.RightMargin,
    tkRight,
    tkUnderline );
// select body again
WPRichText1.ActiveText := WPRichText1.BodyText;
end;

```

Since WPTools V6 You can also use this API:

```

function InputSection(Select: TWPRTFSectionPropsSelect = [wpsec_PageSize, wpsec_Margins]): TWPR
It returns a new section object.

```

This property can be used to retrieve the current section object:

```

function ActiveSection: TWPRTFSectionProps;

```

7.9.8 Syntax Highlighting

Syntax highlighting is useful to avoid logical errors when the user is supposed to edit XML or HTML text in ANSI form or when she or he is editing programming code.

WPTools 5 includes the class [TWPSynEditHighlight](#) which is used as interface to the syntax highlighters included in the open source project SynEdit. Since character attributes, such as "bold" and text color are permanently assigned to the text, it is not possible to deliberately choose text attributes for the text.

WPTools 7 includes a new special syntax highlighting component which dynamically highlights the parts of the text which are detected to be "tokens". It is still possible to apply text attributes.

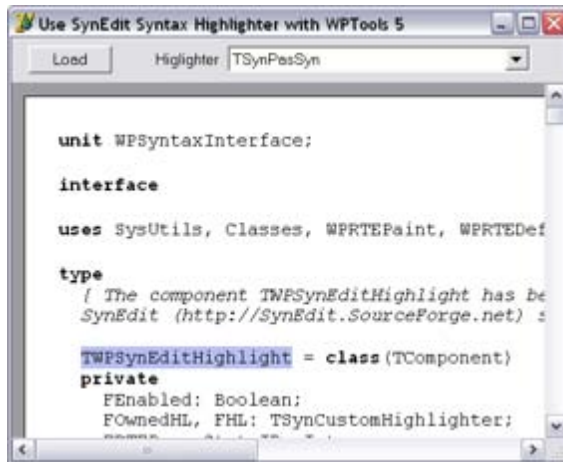
Currently a syntax highlighter for XML and the [special reporting](#) and mail merge tokens are available.

Syntax highlighters can apply character attributes to the text, or, and this is unique, **work non destructive**. Here the character attribute is calculated "on the fly", depending on the context.

7.9.8.1 TWPSynEditHighlight

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 Version 7.

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.

7.9.8.2 TWPCustomSyntax

This class, which is implemented in unit WPSyntaxHighlight implements a basics for a syntax highlighter.

To apply syntax highlighting simply assign a TWPCustomSyntax instance to the property TWPRichText.CustomSyntax. The instance will be automatically freed by the editor.

Example:

```
uses WPSyntaxHighlight;
```

```
HTMLText.CustomSyntax := TWFXMLSyntax.Create(nil);
```

TWPCustomSyntax implements this interface:

Initialize - receives a reference to the "Memo" object in a TWPRichText
procedure Init(Engine: TWPRTFEngineBasis); override;

Called for each paragraph at the start
procedure StartPar(par: TParagraph); override;

Called for each character. It has to update the variable FMode according to the syntax
procedure NextChar(par: TParagraph; CPos: Integer); override;

Called at the end of each paragraph
procedure EndPar(par: TParagraph); override;

Called before the complete text is processed

```
procedure PreProcess(RTFData: TWPRTFDataBlock); override;
```

Called after the complete text was processed

```
procedure PostProcess(RTFData: TWPRTFDataBlock); override;
```

Used to calculate the CharAttr index used at a certain position of a paragraph.

```
function CalcAttr(attr: Cardinal; par: TParagraph): Cardinal; override;
```

Reference to the RTFDataProps object

```
property RTFProps: TWPRTFProps read FRTFProps;
```

7.9.8.3 TWPXMLSyntax

The XML highlighter TWPXMLSyntax descends from TWPCustomSyntax.

It is used to highlight XML and HTML code. It uses a fixed pitch fonts.

It only overrides two methods:

```
constructor TWPXMLSyntax.Create(aOwner: TComponent);  
begin  
    inherited Create(aOwner);  
    FReservedColor := clBlue;  
end;
```

and NextChar which detects the meaning of certain parts of the text. The caller takes care that "FMode" is updated with the mode which was active when the previous paragraph ended.

```
procedure TWPXMLSyntax.NextChar(par: TParagraph; CPos: Integer);  
var C: WideChar;  
    function CC(i: Integer): WideChar;  
    begin if (CPos + i < 0) or (CPos + i >= par.CharCount) then Result := #0 else Result := par.  
begin  
    if FRTFProps <> nil then  
    begin  
        C := par.CharItem[CPos];  
        if not (FMode in [wpsynMLComment, wpsynComment, wpsynReserved, wpsynString]) then  
        begin  
            if (C = '&') then FMode := wpsynChar  
            else if (C = '<') then  
            begin  
                if (CC(1) = '!') and (CC(2) = '-') and (CC(3) = '-') then  
                    FMode := wpsynMLComment  
                else FMode := wpsynReserved;  
            end;  
            if C = '>' then Error := true;  
        end  
    else  
    begin  
        if (FMode = wpsynReserved) and (c = #32) then FMode := wpsynString  
        else if (FMode = wpsynString) and (c = '>') then FMode := wpsynReserved;  
        if C = '<' then Error := true;  
    end;  
    inherited NextChar(par, CPos);  
    if (FMode = wpsynMLComment) and (C = '>') and (CC(-1) = '-') then  
        FMode := wpsynNormal  
    else if (FMode = wpsynReserved) and (C = '>') then  
        FMode := wpsynNormal  
    else if (FMode = wpsynChar) and ((C = ';') or (C = #32)) then  
        FMode := wpsynNormal;
```

```

        Ferror := false;
    end;
end;

```

7.9.8.4 TWPXMLRTFSyntax

This class highlights XML tags in the text but does not apply permanent changes to the formatting.

7.9.8.5 TWPFielSyntax

This class highlights field tokens marked by certain strings. It does **not** apply permanent changes to the text.

This class publishes the properties

FieldStart and FieldEnd. They are initialized with '<<' and '>>'.

7.9.8.6 TWPFielBandSyntax

This class highlights field and band tokens marked by certain strings. It does **not** apply permanent changes to the text. It is inherited from TWPFielSyntax.

Band tokens start with the sign :, groups with #. Only groups may be nested.

This class publishes the properties

FieldStart and FieldEnd which are initialized with '<<' and '>>'.

In addition to TWPFielSyntax it also publishes BandChar and GroupChar which are initialized with ':' and '#'.

7.9.9 TextObjects

7.9.9.1 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;
  TXTObj: TWPTextObj; var PrintString: WideString; var WidthInPix,
  HeightInPix: Integer; var PaintObject: TWPTextObj; Xres, Yres: Integer);
begin
  if TXTObj.Name='CHANGEME' then
  begin
    PrintString := 'more...';
  end;
end;

```

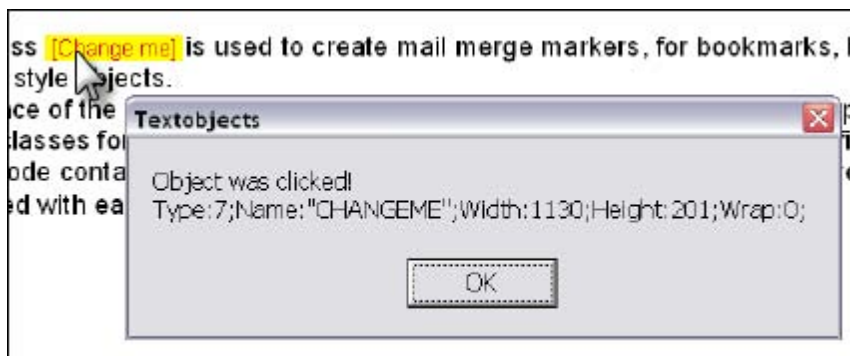
This will display the text object with the text "more..." using the attributes defined for that object.

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

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

The **TWPTextObj** class ➡ is used to
 embedded span style objects.
 In all cases an instance of the **TWPT**

7.9.10 Add PDF Export

Using our Product [wPDF](#) it is very easy to implement PDF export from the WPTools editor.

wPDF is not installed as separate product when used together with WPTools but linked with the WPTools 7 design package.

wPDF can also be used independently, in contrast to many competing products it is a EMF to PDF converter and is as such compatible to most drawing code, wether it uses a TCanvas or HDC handle.

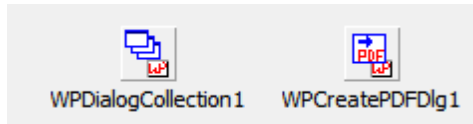
Instead please open the file **WPINC.INC** and activate the \$define **WPPDFEX**

```
{ $DEFINE WPPDFEX } // = link in wPDF, our PDF addon
```

then do a "Build All" with the wptools package. Of course the library path must also list the

wPDF directory.

To bring life to the PDF export button implemented by the TWPToolbar simply drop a component of the type **TWPCreatePDFDlg** on the form, link it with the WPDIALOGCollection (property SaveAsPDF) and, if the WPDIALOGCollection had been already linked to the TWPRichText the toolbutton will show the PDF export dialog.



Alternatively it is also possible to create the PDF export dialog in code and use the toolbar event to launch it:

```
uses ... WPToPDFDlg ...
```

```
procedure TForm1.WPToolBar1IconSelection(Sender: TObject;
  var Typ: TWpSelNr; const str: String; const group, num, index: Integer);
var dlg : TWPCreatePDFDlg;
begin
  if (typ=wptIconSel) and (group=WPI_GR_PRINT) then
  begin
    if num=WPI_CO_PDFExportDialog then
    begin
      try
        dlg := TWPCreatePDFDlg.Create(Self);
        dlg.Filename := WPRichText1.LastFileName;
        dlg.EditBox := WPRichText1;
        dlg.Execute;
      finally
        dlg.Free;
      end;
    end;
  end
  else
  ....
```

In case You do not want to use the dialog, you can export from a TWPRichText directly by dropping the **TWPPDFExport** component. Assign the WPRichText using the property Source and set the filename. Call Export to create the PDF.

If you use Export right after a LoadFromStream or LoadFromFile you need to call WPRichText.ReformatAll(false, false) to format the text. Otherwise the WPRichText will format the text, when the application is idle.

Of course it also makes sense to do this fully in code, especially when the export is not done on a form or the TWPRichText has been created dynamically:

```
uses ..., WPPDFWP, WPRTEDefs, WPCTRMemo, WPCTRRich;

procedure TForm1.ExportFromWPTools(Sender: TObject);
```

```
var pdf : TWPPDFExport;  
begin  
  pdf := TWPPDFExport.Create(nil);  
  pdf.Source := WPRichText1;  
  try  
    pdf.FileName := 'c:\temp\wp5out.pdf';  
    pdf.Print;  
  finally  
    pdf.Free;  
  end;  
end;
```

7.10 M) Appendix

7.10.1 MIME Import / Export

WTools 7 includes the unit "WPIOMime", It implements a read and a writer for MIME encoded HTML Data.

Info: The WTools Demo does not include this unit.

The interface unit requires the powerful HTTP library Arart Synapse. It is not included.
<http://synapse.ararat.cz/>

7.10.1.1 Reader / Writer

To activate MIME support please add the unit WPIOMime to the uses clause. (C++Builder requires #pragma link).

Now it is possible to load *.MHT, *.MSG and *.EML files. These file extensions automatically trigger the MIME reader and writer classes due to the class methods.

```
class function TWPMimeReader.UseForFilterName(const Filtername: string): Boolean;  
begin  
  Result := (CompareText(Filtername, 'TWPMimeReader') = 0) // not "inherited"  
  or (CompareText(Filtername, 'MIME') = 0)  
  or (CompareText(Filtername, 'MHT') = 0)  
  or (CompareText(Filtername, 'MSG') = 0)  
  or (CompareText(Filtername, 'EML') = 0);  
end;
```

and

```
class function TWPMimeWriter.UseForFilterName(const Filtername: string): Boolean;  
begin  
  Result := (CompareText(Filtername, 'TWPMimeWriter') = 0) // not "inherited"  
  or (CompareText(Filtername, 'MIME') = 0)  
  or (CompareText(Filtername, 'MHT') = 0)  
  or (CompareText(Filtername, 'MSG') = 0)  
  or (CompareText(Filtername, 'EML') = 0);  
end;
```

The writer supports the format options:

- nohtml, do not save HTML text
- noplain, do not save PLAIN text

The MimeReader initializes and fills the property `TwPCustomTextReader.CopyOfBodyData` with a copy of the HTML text which is loaded as body. The data can be used to display the HTML source code ([example](#)).

7.10.1.2 Demo

Instead of creating a new demo, we decided to enhance the [WebBrowser application](#) and add 2 buttons, "Load MIME" and "SaveMime".

First we need to add `WPIOMime` to the uses clause:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, WPRTEdefs, WPCTRMemo, WPCTRRich, ComCtrls,
  ExtCtrls, wphttpget_unit, wpobj_image, WPIOMime;
```

To save a web archive or message file we use

```
procedure TWPWebBrowser.SaveMIMEClick(Sender: TObject);
begin
  with TSaveDialog.Create(self) do
    try
      Filter := 'Webarchives (*.MHT)|*.MHT|Messages (*.MSG,*.EML)|*.MSG,*.EML';
      DefaultExt := 'MHT';
      Caption:= 'Save as MIME data';
      if Execute then
        begin
          // Web Archive
          if SameText( ExtractFileExt( FileName ), '.mht' ) then
            WPRichText1.SaveToFile(FileName, false, 'MIME-noplain')
          else WPRichText1.SaveToFile(FileName);
        end;
      finally
        Free;
      end;
    end;
end;
```

To load a MSG or MHT file we use

```
procedure TWPWebBrowser.LoadMIMEClick(Sender: TObject);
begin
  with TOpenDialog.Create(self) do
    try
      Filter := 'Webarchives (*.MHT)|*.MHT|Messages (*.MSG,*.EML)|*.MSG,*.EML';
      DefaultExt := 'MHT';
      Caption:= 'Load MIME data';
      if Execute then
        begin
          WPRichText1.SaveToFile(FileName, true);
        end;
      finally
        Free;
      end;
    end;
end;
```

end;

7.10.2 HTTP Interface

WPTools 7 includes the unit "wphttpget_unit", It implements a set of routines to load text, image data and style sheets over HTTP connections.

The interface unit requires the powerful HTTP library Arart Synapse. It is not included.

<http://synapse.ararat.cz/>

Info: The WPTools Demo does not include this unit.

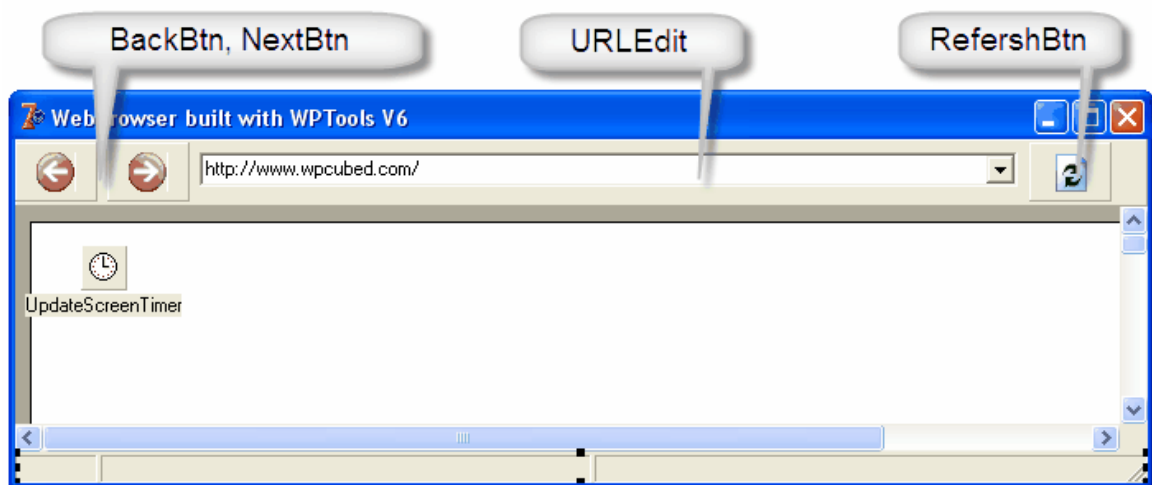
The interface is attached to the WPTools Kernel using function pointers. This makes it possible to separate the HTTP support from the rest of the application and is an effective mean to avoid any security risks through open HTTP connection points.

In this demo (see directory Demos\HTTPGet) we discuss how to implement a webbrowser like application, including history with forward and backwards buttons.

The HTTP interface can be temporarily disabled by setting the boolean variable wphttp_Disable = true.

7.10.2.1 Form Setup

The Form is simple:



Also required are 4 variables and 2 functions. Most of the code will be inserted into event handlers:

```
public
  history: TStringList;
  historypos: Integer;
  pendingload: Boolean;
  procedure Load(url: string; Update: Boolean);
  procedure UpdateButtons;
end;
```

In the Form.OnCreate event we insert

```
procedure TWPWebBrowser.FormCreate(Sender: TObject);
begin
```

```

history := TStringList.Create;

WPRichText1.AsWebpage := [wpFormatAsWebpage];
WPRichText1.OneClickHyperlink := true;
  // No Table Resize
WPRichText1.EditOptions := [wpActivateUndo, wpActivateUndoHotkey];

  // Hot Hyperlinks
WPRichText1.HyperlinkTextAttr.HotEffect := wpeffTextColor;
WPRichText1.HyperlinkTextAttr.HotEffectColor := clBlue;

  // Limit the used fonts
WPRichText1.RTFData.RTFProps.PreselectedFonts.Assign( Screen.Fonts );
WPRichText1.RTFData.RTFProps.PreselectedFonts.Add('sans-serif=Frutiger');
end;

```

The property `PreselectedFonts` is important for the CSS reader.

It makes it possible to select one font alternative in cases multiple are offered in a comma separated list.

In the `Form.OnDestroy` event we insert

```

procedure TWPWebBrowser.FormDestroy(Sender: TObject);
begin
  history.Free;
end;

```

7.10.2.2 Unit Initialization

To update the status bar we need to create a global method. This method will be called when data is loaded over HTTP connections. It also initializes a timer on the form to update the screen and to clear the status bar.

```

procedure do_wphttp_Notify(code: Integer; text: PAnsiChar); stdcall;
begin
  if WPWebBrowser <> nil then
    begin
      if code = 1 then //Load HTML
        begin
          WPWebBrowser.StatusBar1.Panels[1].Text := StrPas(text);
          if WPWebBrowser.historypos < WPWebBrowser.history.count then
            WPWebBrowser.history[WPWebBrowser.historypos] := StrPas(text);
          end
        else if code = 2 then // Called Synchronized !
          begin
            WPWebBrowser.StatusBar1.Panels[2].Text := StrPas(text);
            WPWebBrowser.UpdateScreenTimer.Enabled := TRUE;
            WPWebBrowser.pendingload := TRUE;
          end;
        end;
      end;
    end;

```

This is the timer method

```

procedure TWPWebBrowser.UpdateScreenTimerTimer(Sender: TObject);
begin
  if not pendingload then

```

```

begin
  WPWebBrowser.StatusBar1.Panels[2].Text := '';
  UpdateScreenTimer.Enabled := FALSE;
end;
WPRichText1.Repaint;
pendingload := FALSE;
end;

```

The method `do_wphttpNotify` and the 4 methods from unit `wphttpget_unit` are attached to the WPTools engine using function pointers. The pointers are set in the initialization section of the unit:

initialization

```

// HTTP functions
WPRTEDefs.wphttp_Init := @wphttpget_unit.wphttp_Init;
WPRTEDefs.wphttp_Exit := @wphttpget_unit.wphttp_Exit;
WPRTEDefs.wphttp_get := @wphttpget_unit.wphttp_get;
WPRTEDefs.wphttp_copydata := @wphttpget_unit.wphttp_copydata;
wphttp_Init_param := 'WPCubed_GmbH_HTTP1';

// Also initialize wphttp_Notify
WPRTEDefs.wphttp_Notify := do_wphttp_Notify;

end.

```

7.10.2.3 User Action and History Management

The history uses a string list and an index into this list.

Event for both buttons.

```

procedure TWPWebBrowser.BackBtnClick(Sender: TObject);
begin
  if Sender = BackBtn then // Click on <--
    dec(historypos)
  else inc(historypos); // Click on -->
  Load(history[historypos], false);
end;

```

and the method which updates the enabled state

```

procedure TWPWebBrowser.UpdateButtons;
begin
  BackBtn.Enabled := (historypos > 0) and (history.Count > 0);
  NextBtn.Enabled := (historypos < history.Count - 1);
end;

```

The Page is loaded when clicking "Refresh" or when clicking on a hyperlink or when pressing enter in the URL edit field.

```

procedure TWPWebBrowser.RefrershBtnClick(Sender: TObject);
begin
  Load(URLEdit.Text, false);
end;

```

```

procedure TWPWebBrowser.WPRichText1HyperLinkEvent(Sender: TObject; text,
  url: string; IgnoredNumber: Integer);
begin
  inc(historypos);
  Load(url, true);
end;

```

```

procedure TWPWebBrowser.URLEditKeyPress(Sender: TObject; var Key: Char);
begin
  if Key = #13 then
    begin
      Load(URLEdit.Text, true);
      Key := #0;
    end;
end;

```

7.10.2.4 Load Method

The method load is responsible to load documents and also images. For unsupported file extensions it calls MessageBeep and exits. It does not yet preload the data and check it for errors.

```

procedure TWPWebBrowser.Load(url: string; Update: Boolean);
var s: string;
    obj: TWPOImage;
    load_the_text: Boolean;
begin
  load_the_text := true;
  s := lowercase(ExtractFileExt(url));
  if (s = '.rtf') or (s = '.txt') or (s = '.doc') then
    begin
      // Standard Mode for Documents
      WPRichText1.AsWebpage := [];
    end
  else
    if (s = '.jpg') or (s = '.gif') or (s = '.png') or (s = '.jpeg') then
      begin
        // Mode for images
        obj := TWPOImage.Create(WPRichText1.RTFData);
        if obj.LoadHTTPFromThread(nil, WPRichText1.RTFData, WPRichText1.RTFData.HTTPPrepareURL(url)) then
          begin
            WPRichText1.AsWebpage := [];
            WPRichText1.Clear;
            WPRichText1.InputObject(obj);
            load_the_text := false;
          end else
            begin
              obj.Free;
              MessageBeep(0);
              exit;
            end;
          end
        else // Ignore data right away
          if (s = '.exe') or (s = '.zip') or (s = '.pdf')
            or (s = '.rar') or (s = '.chm') or (s = '.hlp')
          then
            begin // Unsupported extensions
              MessageBeep(0);
              exit;
            end
          else
            begin // Mode for websites
              // TODO - we should load now preload the data and check if valid
              WPRichText1.AsWebpage := [wpFormatAsWebpage];
            end;
            // Update Statusbar
            URLEdit.Text := url;
            StatusBar1.Panels[1].Text := url;
            // Update History NOW - otherwise do_wphhttp_Notify cannot update the filename correctly
            if Update then

```

```

begin
  while history.Count > historypos do history.Delete(history.Count - 1);
  historypos := history.Count;
  history.Append(url);
  UpdateButtons;
end;
// Do we need to load text (maybe we loaded images)
if load_the_text then
begin
  WPRichText1.Clear;
  WPRichText1.LoadFromFile(url);
end;
end;
end;

```

Notes:

Most of the work is done in WPRichText.LoadFromFile - here WPTools 7 detects http urls and uses the http functions to load the data.

Images are loaded multithreaded - so the page appears quickly.

The new TWPObject class implements the method LoadHTTPFromThread (it will not work with threads though, if the last parameter is false).

Here we also activate the special HTML formatting mode, or disable it for regular RTF documents.

7.10.2.5 Webbrowser with WPTools

Although WPTools 7 is not meant to replace a specialized HTML component, You can build a pretty decent HTML viewer with it.



For animated GIFs you first need "GifImage", a free GIF library. Please see unit wpObj_Image.PAS for more infos and how to activate the GIF support.

You then need a timer on the form and with it You call

```
procedure TWPWebBrowser.GIFTimerTimer(Sender: TObject);
begin
    WPRichText1.RefreshAniImages;
end;
```

Don't forget to set the ViewOption "wpUseOwnDoubleBuffer"

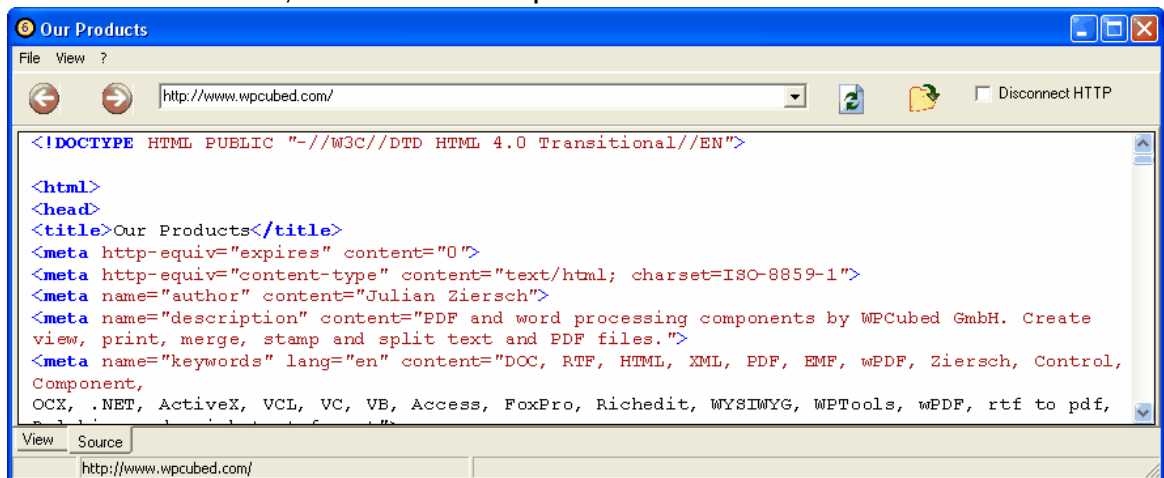
```
WPRichText1.ViewOptions := WPRichText1.ViewOptions + [wpUseOwnDoubleBuffer];
```

7.10.2.6 Source View

It is possible to capture the HTML source code and display it in a second editor:

Here we have two tabs, the first shows the TWPRichText which displays the formatted text, the other shows the source code and uses the XML syntax highlighting.

The editors are individual, edits in one will not update the other editor.



The second editor is initialized like this:

```
procedure TWPWebBrowser.FormCreate(Sender: TObject);
begin
    ...
    // Update the Source view
    WPRichText1.RTFData.AfterLoadFromStream := WPRichText1LoadFromStream;
    // uses WPSyntaxHighlight:
    SourceView.CustomSyntax := TWPXMLSyntax.Create(nil);
    ...
end;
```

The text is assigned in the event **TWPRTFDataCollection.AfterLoadFromStream**. The handler uses this code:

```

procedure TWPWebBrowser.WPRichText1LoadFromStream(
  RTFData: TWPRTFDataCollection;
  Stream: TStream;
  Reader: TWPCustomTextReader;
  OnlyBodyText: Boolean;
  var LoadedText: TWPRTFDataBlock);
begin
  if not OnlyBodyText then
    begin
      Stream.Position := 0;
      if Reader.CopyOfBodyData<>nil then
        // HTML extracted from MIME data
        SourceView.LoadFromStream(Reader.CopyOfBodyData, 'ANSI', true)
        // Original Stream
      else SourceView.LoadFromStream(Stream, 'ANSI', true);
      Caption := WPRichText1.RTFVariables.Strings['Title'];
    end;
  end;

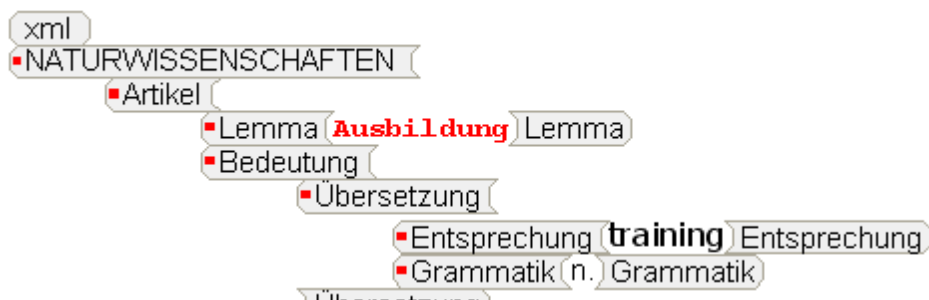
```

Here we use the property `CopyOfBodyData`. This memory stream is only filled by the [MIME](#) reader to capture the HTML body source. The normal stream would show the complete message text, including all header lines and attachments.

7.10.3 XML editor mode

WPTools 7 supports a special XML editing mode.

A loaded XML text will be displayed as:



for this the XMLTAGS reader must be used.

```
WPRichText1.LoadFromFile( filename , true, 'XMLTAGS-utf8' );
```

and the FormatOptions initialized as follows:

```

WPRichText1.FormatOptions := WPRichText1.FormatOptions + [wpShowSPANCode
WPRichText1.FormatOptionsEx2 := WPRichText1.FormatOptionsEx2 + [wpfCode

```

`wpfCodeObjectAsXMLTags` creates the tag boxes shown above.
`wpfCodeObjectCheckParStyles` makes the editor look up paragraph styles which have the same name as the tags which include the text. In this case the paragraph style collection uses the CSS definition

```
Lemma{font-family:'Courier New';font-weight:bold;color:red}
Entsprechung{font-family:'Tahoma';font-size:13.00pt;}
```

which was assigned to the editor using:

```
WPRichText1.LoadCSSheet( css_string );
```

... but there is more ...

in case You have the WPTools 7 Premium edition, You can also work with XML schemas.

from the tags in the XML schema it is possible to create a popup menu with items, which can be inserted at a certain position in the editor.

To make this easy the premium edition includes the unit WPXMLSchema which implements the class TWPXMLSchmema.

You need to create it like this:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Schema := TWPXMLSchmema.Create(Self);
    Schema.OnPopupMenuClick := DoPopupMenuClick;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    Schema.Free;
end;
```

DoPopupMenuClick is a method which should handle the click event on one popup menu item.

```
procedure TForm1.DoPopupMenuClick(fMenu: TXMLMenuItem; fXMLSchema: TWPXMLSchmema; fXMLTag: TWPXMLTag);
var men: TXMLMenuItem;
begin
    men := fMenu.XMLParent;
    while men <> nil do
        begin
            WPRichText1.InputCode(wpobjCode, men.XMLTag.nameparam, '', [wpinpWrapSelectedText, wpinpNewParagraph],
            men := men.XMLParent;
        end;
        WPRichText1.InputCode(wpobjCode, fXMLTag.nameparam, '', [wpinpWrapSelectedText, wpinpNewParagraph],
        WPRichText1.SetFocus;
    end;
```

The popup menu object is created on the form and assigned to the WPRichText PopupMenu property. Then in event PopupMenu1Popup this code is placed:

```
var bb: Boolean;

procedure TForm1.PopupMenu1Popup(Sender: TObject);
var a: TWPXMLSchemaTag; pt: TPoint;
begin
    if not bb then
        try
            bb := true;
            PopupMenu1.Items.Clear;
```

```

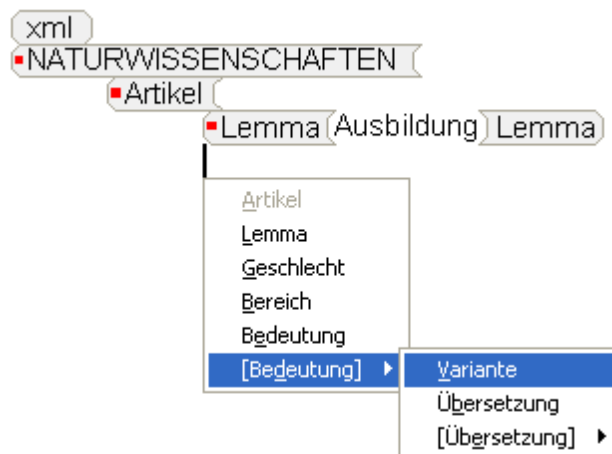
a := Schema.LocateContext(
  WPRichText1.ActivePar,
  WPRichText1.ActivePosInPar);
if a <> nil then
  a.FillPopupMenu(PopupMenu1, 0);
WPRichText1.GetParXYBaselineScreen(WPRichText1.ActivePar,
  WPRichText1.ActivePosInPar, pt.x, pt.y);
pt := WPRichText1.ClientToScreen(pt);
PopupMenu1.Popup(PopupMenu1.PopupPoint.X, pt.y + 4);
finally
  bb := false;
end;
end;
end;

```

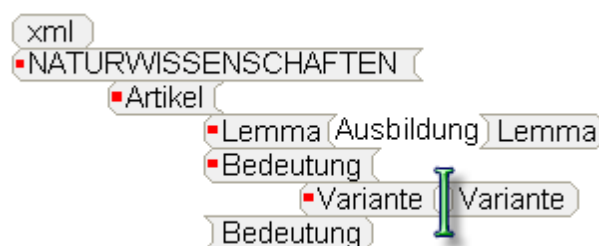
Of course the menu can be also created a bit differently or other items can be added.

In action the popup menu can look like this:

1) press the context menu key on the keyboard:



2) select the item and press enter



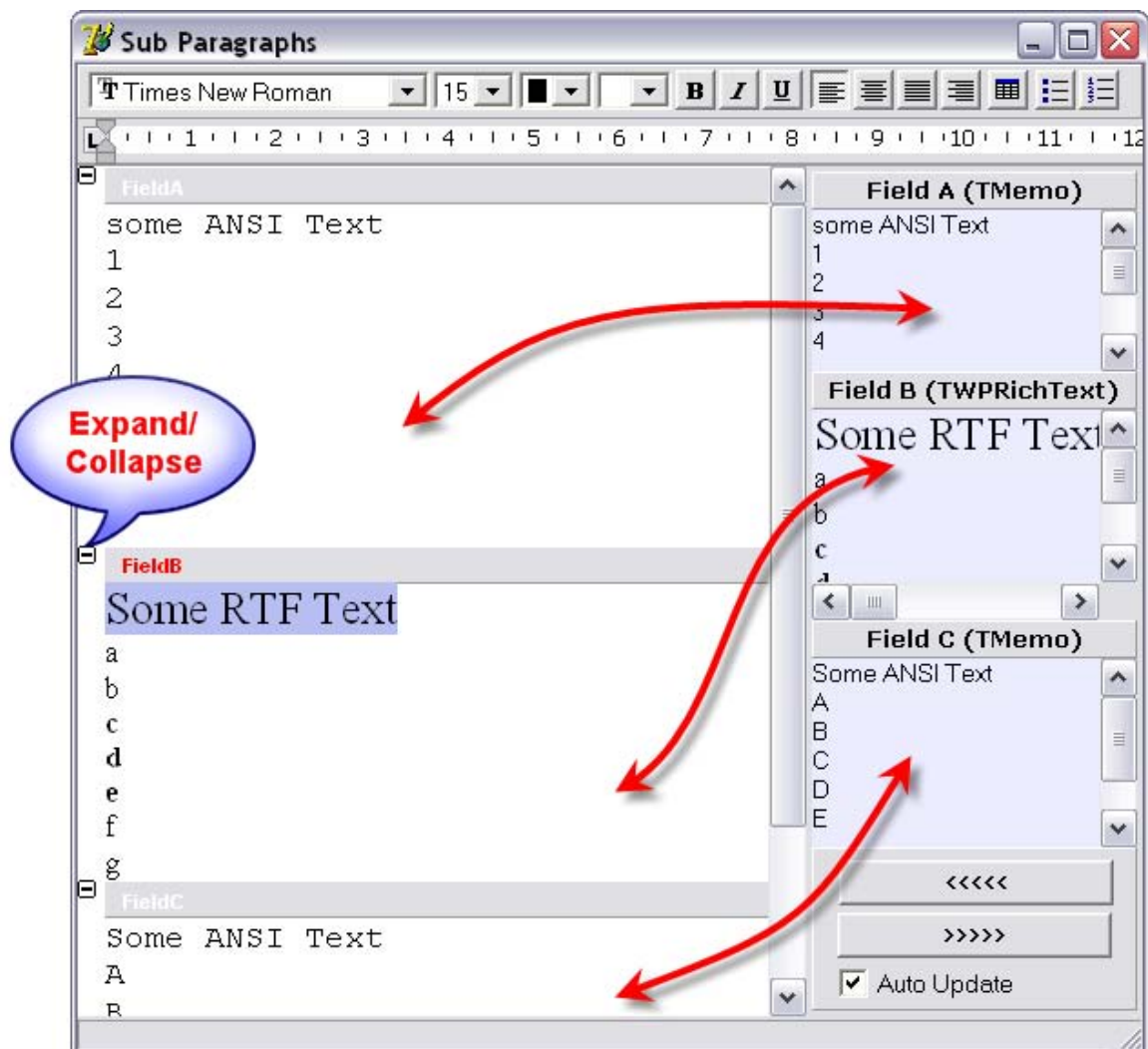
7.10.4 Work with sub paragraphs

WPTools Version 7 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 thinkable 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', '', [wplloadpar_AsChildrenPar]); // 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;

```

7.10.5 Use "External Pages"

What are "external pages" ?

This are pages which are displayed by WPTools Version 7 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.

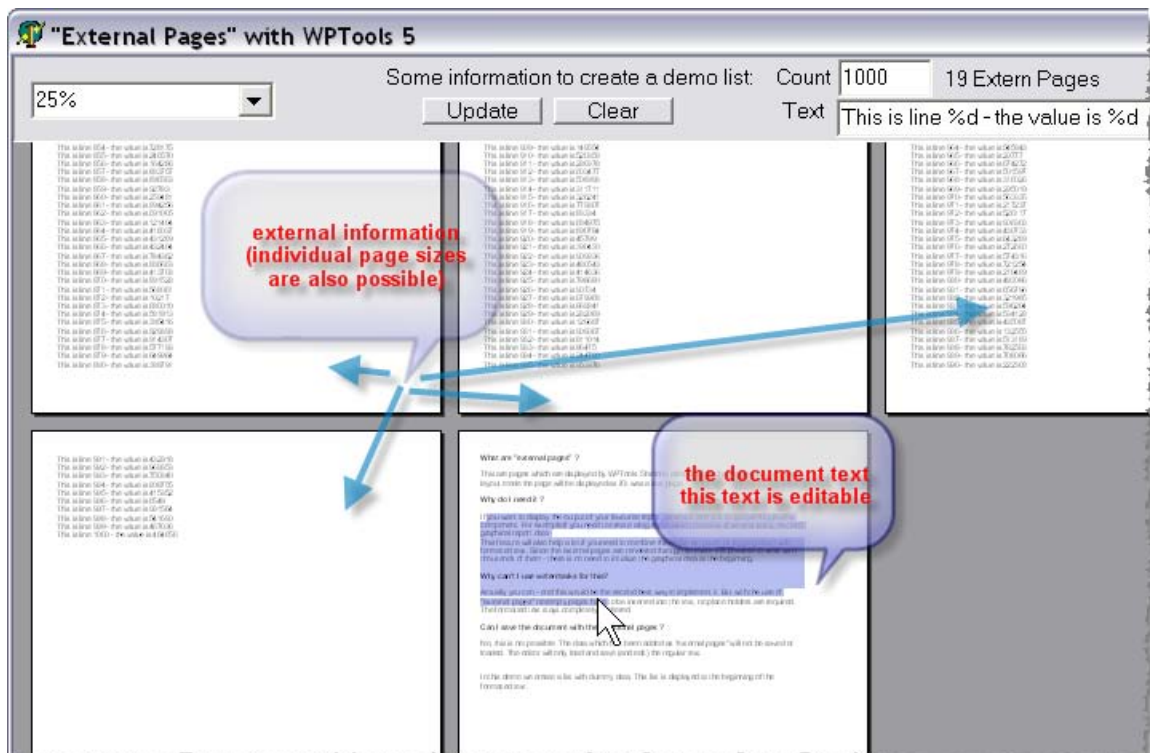
Why can't I use watermarks for this?

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 stays
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 TWPEExternalP.WPRichText1PaintExternPage(Sender: TObject;
  RTFEngine: TWPRTFEnginePaint; prCanvas: TCanvas; xoff, yoff: Integer;
  r: TRect; PaintPageNr: Integer; ExternPageRef: TObject; DestXRes,

```

```

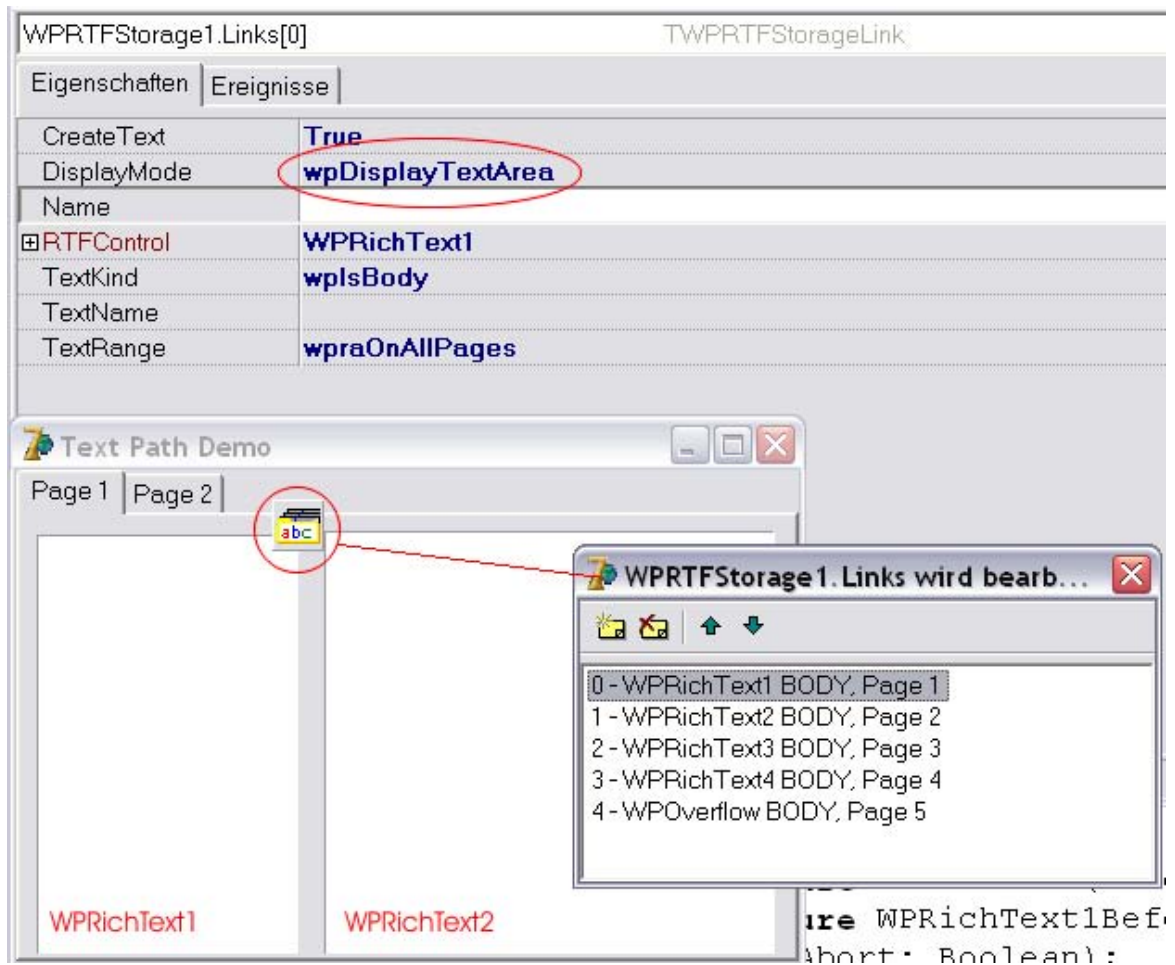
DestYRes: Integer);
begin
prCanvas.StretchDraw( r, ExternPageRef as TGraphic );
end;

```

7.10.6 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;

```

7.10.7 Bookmarks

The bookmarks of WPTools Version 7 are very powerful. They can be nested and several functions make it easy to work with the marks.

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

```

[TEXT<[WHATS_NEW<What's new in WPTools Version 5?>WHATS_NEW]
[PART_ONE<[PAR1<This version has been improved in many aspects. It is faster
(especially tables), reliable and implements innovative programming concepts. >PAR1]

```

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 inside of the new bookmark markers }
function BookmarkInput(const Name: string; PlaceCursorBetweenTags: Boolean =
FALSE): TWPTTextObj;
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: TStringList; FromAllBlock: Boolean = FALSE);
overload;
procedure BookmarkGetList(list: TWPTTextObjList; 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: TWPTTextObj): 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: TWPTTextObj;
function BookmarkFirstInPar(par: TParagraph): string;
function BookmarkAtParLin(par: TParagraph; pos_in_par: Integer): TWPTTextObj;
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;
  TXTObj: TWPTTextObj; var PrintString: WideString; var WidthInPix,
  HeightInPix: Integer; var PaintObject: TWPTTextObj; Xres, Yres: Integer);
begin
  if not ShowBookmarkCodes.Checked then exit;
  if TXTObj.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 := TXTObj;
    PaintObject.OnPaint := OnPaintObject;
    WidthInPix := RefCanvas.TextWidth(TXTObj.Name+'<'+#32);
  end;
end;
```

```

procedure TForm1.OnPaintObject(Sender : TWTextObj;
  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 : TWTextObjList;
  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;

```

7.11 N) PDF export with wPDF

Used alone, with your own component or linked with WPTools, [wPDE](#) instantly creates PDF files at high speed.

wPDF supports

- Developer Express(tm) EXPRESS PrintingSuite, (example included)
- ReportBuilder, (interface included)
- ACE Reporter, (interface included)
- RichEdit, (example included)
- RAVE Report, (interface included)

- HTMLView,
- FAST Report 2, (interface included)
- FAST Report 3+4, (interface included)
- QuickReport, (interface included)
- WPTools 4 (incl. links and bookmarks, component included) ,
- WPTools 5 (incl. links and bookmarks, component included) ,
- WPTools 7 (incl. links, fields and bookmarks, component included) ,
- WForm, (example included)
- List&Label

and most other components which can create a metafile or draw to a HDC / Canvas.
You always can combine the output of different components into one PDF file!

The integration with WPTools extremely tight. So, other than just exporting text and images these features are supported:

- Export hyperlinks
- Export Bookmarks
- Export JPEG data without additional compression
- Create PDF tags (Also known as "marked PDF" - this is a prerequisite for PDF/A compliancy. Tagged PDF Documents can be better converted to formatted text. Our own "PDF to RTF" converter will even recreate table structures!)
- new in WPTools 7: create edit fields ([Example ...](#))
- new in WPTools 7: create checkbox fields ([Example ...](#))
- new in WPTools 7: embed data stored in a special object instance ([Example ...](#))

7.11.1 Export to PDF

This is the most compact code to export the RTF or HTML text from a WPTools Editor to PDF:

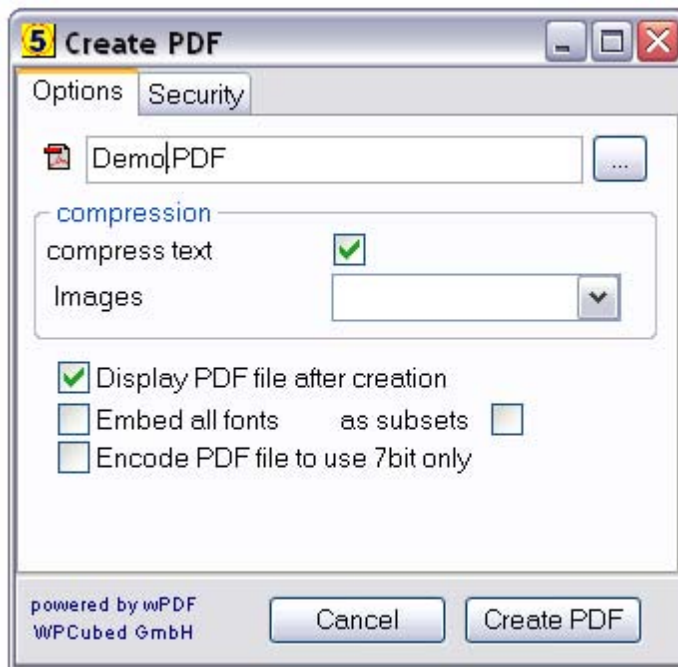
```
uses ..., WPPDFWP, WPRTEDefs, WPCTRMemo, WPCTRRich;

procedure TForm1.ExportFromWPTools(Sender: TObject);
var pdf : TWPPDFExport;
begin
  pdf := TWPPDFExport.Create(nil);
  pdf.Source := WPRichText1;
  try
    pdf.FileName := 'c:\exported_rtf';
    pdf.Print;
  finally
    pdf.Free;
  end;
end;
```

The TWPPDFExport component can be alternatively dropped on the form and the properties can be set in the IDE. Then only one line of code is required: WPPDFExport1.Print.

7.11.2 Export using dialog

You can use the provided PDF creation dialog to create a full featured PDF file with 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;
```

7.11.3 Export using PaintRTFPage()

Alternatively this simple code can be used to export PDF files from WPTools Version 7.

The drawing code used by WPTools Version 7 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 := MulDiv(WPRichText1.Memo._PaintPages[i].WidthTw, WPScreenPixelsPerInch, 1440);
    h := MulDiv(WPRichText1.Memo._PaintPages[i].HeightTw, WPScreenPixelsPerInch, 1440);
    if (w=0) or (h=0) then
    begin
      w := Round( WPRichText1.Memo.PaintPageWidth[i] / WPRichText1.Memo.CurrentZooming );
      h := Round( WPRichText1.Memo.PaintPageHeight[i] / WPRichText1.Memo.CurrentZooming );
    end;
    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, [wppInPaintForwPDF] );
    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;

```

```
end;
end;
```

7.11.4 Create edit / memo fields

Using WPTools 7 and wPDF V3 ist is now possible to create text fields in PDF files.

Example:

This is a edit field:

Create a simple edit field

```
with WPRichText1.InputTextFieldName('FORMEDITFIELD') do
begin
    Source := FileName.Text;
    Params := SFieldText.Text + '@@HINT@@" + SHintText.Text;
end;
```

Create a edit field based on a TWPObject class:

Here it is also possible to create multi line fields.

```
var obj : TWPObj;
begin
    if Multiline.Checked then
        obj := WPRichText1.TextObjects.InsertClass('TWPOEditControl', 360*5, 260*5 )
    else obj := WPRichText1.TextObjects.InsertClass('TWPOEditControl', 360*5, 260 );
    if obj.IsImage then
        begin
            obj.ObjRef.ObjName := AFieldName.Text;
            (obj.ObjRef as TWPOEditControl).Text.Text := TextEdit.Text;
            (obj.ObjRef as TWPOEditControl).Hint := HintEdit.Text;
            (obj.ObjRef as TWPOEditControl).PDFFont := TWPOEditControlPDFFont( FontSelect.ItemIndex );
            if not AutoSize.Checked then
                (obj.ObjRef as TWPOEditControl).PDFOptions :=
                    (obj.ObjRef as TWPOEditControl).PDFOptions - [wpecAutosizeFont];
            (obj.ObjRef as TWPOEditControl).Multiline := Multiline.Checked;
            if Multiline.Checked then (obj.ObjRef as TWPOEditControl).FontSize := 10;
        end;
    end;
```

7.11.5 Create a check box field

Using WPTools 7 and wPDF V3 ist is now possible to create check boxes in PDF files.

Example:

This is a check box: ☒

This box was created with this code:

```
with WPRichText1.InputTextFieldName('FORMCHECKBOX') do
begin
    Source      := FieldName.Text;
    Params      := 'true';    //'false'
end;
```

7.11.6 Create embedded data objects

Using WTools 7 and wPDF V3 it is now possible to add an object to the PDF file which holds embedded data.

Example:

This is a container  Dies ist ein Test

This feature uses the class TWPODataContainer. It is a descendant of TWPOImage and so able to display images.

The embedded data is loaded by AddDataFile or AddDataStream.

```
function AddDataFile(Filename: WideString;
    fsmode: TPDFEmbeddedFileElementFS = wpembDefault): string;

function AddDataStream(data: TStream;
    fsmode: TPDFEmbeddedFileElementFS;
    compressmode: TPDFEmbeddedFileElementCompress;
    const ext_or_subtype: string): string;
```

Example code:

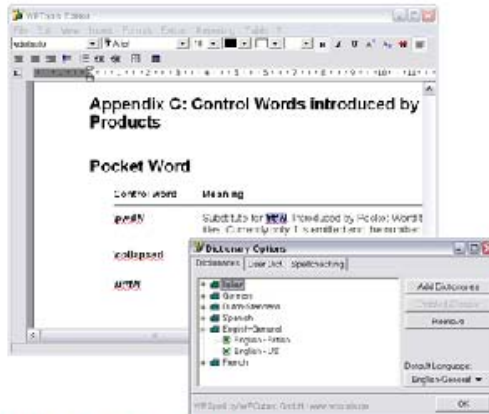
```
var obj : TWPOTextObj;
begin
    obj := WPRichText1.TextObjects.InsertClass('TWPODataContainer', 360, 360 );
    if obj.IsImage then
    begin
        obj.Mode := obj.Mode + [wpobjSizingDisabled];
        // obj.ObjRef.LoadFromFile('c:\a.bmp');
        obj.ObjRef.AssignBitmap(Image1.Picture.Graphic);

        (obj.ObjRef as TWPODataContainer).AddDataFile(Edit1.Text, wpembDefault);
        (obj.ObjRef as TWPODataContainer).Icon := wpemTag; // wpemGraph;
        (obj.ObjRef as TWPODataContainer).AddParam('Contents', 'Dies ist eine Datei');
        (obj.ObjRef as TWPODataContainer).AddParam('Title', 'Julian Ziersch');
    end;
```

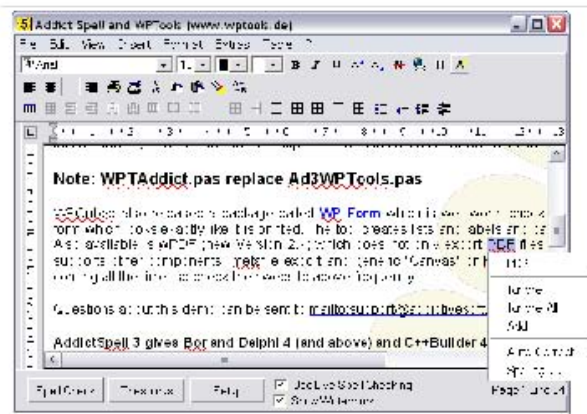
7.12 O) Adding Spellcheck

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 [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

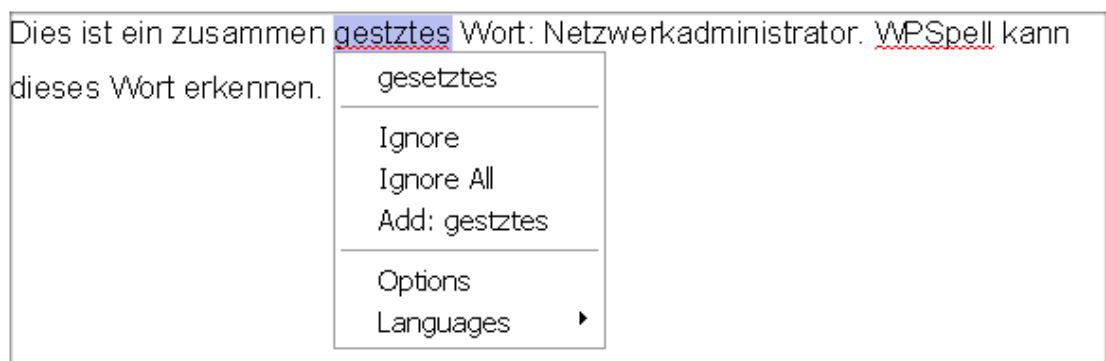
- Spellcheck while typing
- WPSpell has been especially tailored to work with WPTools.
- Traditional spellcheck dialog
- Support for spellcheck during input (curly underlines + popup dialog)
- With WPTools Version 7: instant update of spellcheck markers when switching languages
- Use multiple dictionaries
- 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 compound 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:



7.12.1 Use WPSpell

WPSpell is an addon to WPTools.

1) Please add the required units to the uses clause

```
uses WPSpell_link, WPSpell_Controller, WPSpell_OptForm, WPSpell_StdForm
```

2) Create a TWPSpellController (you can also drop the component on the form) and set the properties. (Note: This step is not required if you are using the DEMO version since here the SpellController is provided by a DLL - the registered version does not use a DLL)

in TForm we need a variable:

```
FSpellController : TWPSpellController;
```

This variable is initialized in the event OnCreate

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  FSpellController := TWPSpellController.Create(self);
  FSpellController.PersistencyMode := wpUseRegistry;
  FSpellController.Active := TRUE;
  FSpellController.LoadSetup(false);
```

```
// UpdateLanguges; - this procedure fills a combobox
end;
```

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
    FSpellController.AutoSaveSetup;
    FSpellController.Free;
end;
```

3) Switch SpellAsYouGo on and off (*Spell* is a checkbox in this example)

```
procedure TForm1.SpellClick(Sender: TObject);
begin
    if Spell.Checked then WPRichText1.StartSpellCheck(wpStartSpellAsYouGo)
    else WPRichText1.StartSpellCheck(wpStopSpellAsYouGo);
end;
```

4) Add a configure button or menu item

```
procedure TForm1.ConfigureClick(Sender: TObject);
begin
    if FSpellController.Configure then
        begin
            // UpdateLanguges; - if we update a combobox
            if Spell.Checked then
                begin
                    WPRichText1.StartSpellCheck(wpStopSpellAsYouGo);
                    WPRichText1.StartSpellCheck(wpStartSpellAsYouGo);
                end;
            end;
        end;
```

alternatively you can also call:

```
WPRichText1.StartSpellCheck(wpShowSpellCheckSetup);
```

5) If you want to show a combobox with the available dictionaries

```
// This procedure updates the combobox
procedure TForm1.UpdateLanguges;
var i, j : Integer;
begin
    // Checkbox
    Spell.Checked := (FSpellController.OptionFlags and WSPELLOPT_SPELLASYOUGO)<>0;
    // ComboBox
    SpellLanguages.Items.Clear;
    j := -1;
    for i:=0 to FSpellController.LanguageIDCount-1 do
        begin
            if FSpellController.CurrentLanguage=FSpellController.LanguageID[i] then
                j := SpellLanguages.Items.Count;
            SpellLanguages.Items.AddObject(FSpellController.LanguageName[i],
                Pointer(FSpellController.LanguageID[i]));
        end;
    SpellLanguages.ItemIndex := j;
end;
```

```
//this procedure applies a language change
procedure TForm1.SpellLanguagesChange(Sender: TObject);
begin
    if SpellLanguages.ItemIndex>=0 then
        begin
```

```

    FSpellController.CurrentLanguage :=
    Integer(SpellLanguages.Items.Objects[SpellLanguages.ItemIndex]);
    if Spell.Checked then
    begin
        WPRichText1.StartSpellCheck(wpStopSpellAsYouGo);
        WPRichText1.StartSpellCheck(wpStartSpellAsYouGo);
    end;
end;
end;
end;

```

The property **CurrentLanguage** is used to select the current dictionary. You have to use one of the language or language-group IDs which are defined in file WPLanguages.INC

Example:

```

English General = 9,
English UD= 1033,
English British = 2057,
French = 1036,
German = 1031,
Italian= 1040,
Spanish = 1034

```

7.13 K) 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). In contrast to the the plenty reporting applications and tools available already, our reporting engine is based on a word processor. This means the reporting template is just a text document, so is the output.

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



TWPFFormulaInterface

This class creates a link between an editor and an TWPEvalEngine to



TWPSuperMerge

support calculation in text and tables
This is the main class of WPSuperMerge.
It implements the logic to create a new
text from data and template texts.

WPSuperMerge 7 includes powerful [Token to Template Conversion](#). ReportTemplates can be edited using certain plain text tokens.

Order link to the [upgrade](#) WPSuperMerge Standard or WPSuperMerge Standard PRO WPSuperMerge "Bundle"

7.13.1 Comparison to "usual" reporting

1) End user can modify templates

Many reporting tools are very complicated to use or do not offer end user modifications at all.

a) With WPSuperMerge it is possible to load a regular document with added tags for fields (i.e. `<<name>>`) and bands (i.e. `<<#ORDERS>> .... <<#/ORDERS>>`) and WPSuperMerge can convert such a document into a reporting template ([=token to template conversion](#)). Syntax highlighting is possible while editing the reporting template in text form.

b) If the database structure is rather complicated you can use the alternative template architecture. Here the report logic has been split up into two parts.

One part is created and maintained by the developer. It contains the outline of the report with bands which can but do not have to be used, fields which can be used and variables which are calculated while the report is being processed as XML data.

The second part is the report template. It can be edited by the end user (you can of course hide it, too) using the word processor. It is possible to insert fields and variables from the "repository". It is also possible to reset a selected group to its initial state in case it has been messed up. It is saved as document with added proprietary attributes to maintain the bands.

2) No fixed page layouts

Most reporting tools work with page layouts. This means you may place graphics at exact positions on a form and when the report is created it will exactly look as the designed forms.

This approach is often very good, but in some cases you want the text created by the reporter to be edit able. The mentioned approach has a problem here - while it is sometimes possible to create RTF documents with the reporting tool these are not really "edit able" because the text has been broken up into the tiniest pieces - just single text boxes. Usually the RTF export is optimized to be best viewed in MS Word only.

With the WPSuperMerge a text file (RTF or WPT format) is created which allows full editing, including changing of the page size, and other operations which makes the re-pagination of the text necessary.

This means that the reporting feature is optimal if you need to create longer texts, such as contracts. It will be suitable to print lists and invoices. If you need to print forms which look like pre printed paper forms (i.e. TAX forms) you cannot use the WPTools reporting. But in this case you will probably not need sophisticated formatted text.

3) Most RTF attributes usable

Usually the reporting tools only support very limited RTF features, sometimes even justified text is a problem, not to speak of images with text wrapping around, tables and more sophisticated tab stops which use fill signs.

Since the WPTools reporting uses the same text engine all the powerful word processing features can be used in the created report.

4) No external installation

Many reporting tools require to be installed separately, they can be quite cost intensive and the licensing can be restrictive.

7.13.2 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 TWPTTextObj objects. When the text is calculated the paragraph text or the displayed text of the TWPTTextObj 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:

This are the ordered products:

	Product	Price	Amount	net	+VAT	total
1	Cool	1	1	PAR_NET left(2)*left(1) 1,00	PAR_VAT left(1)*0.16 0,16	PAR_TOTAL left(2)+left(1) 1,16
2	Master	862	1	PAR_NET left(2)*left(1) 862,00	PAR_VAT left(1)*0.16 137,92	PAR_TOTAL left(2)+left(1) 999,92
3	Hummer	273	3	PAR_NET left(2)*left(1) 819,00	PAR_VAT left(1)*0.16 131,04	PAR_TOTAL left(2)+left(1) 950,04
4	High Performace	319	1	PAR_NET left(2)*left(1) 319,00	PAR_VAT left(1)*0.16 51,04	PAR_TOTAL left(2)+left(1) 370,04
5	Better	373	2	PAR_NET left(2)*left(1) 746,00	PAR_VAT left(1)*0.16 119,36	PAR_TOTAL left(2)+left(1) 865,36
				2747,00 PAR_NET	439,52 PAR_VAT	3186,52 PAR_TOTAL

Please pay **3186,52** TWPTextObj

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 TWPFomulaInterface. 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 TWPFomulaInterface.**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.

NR	VALUE	Header Row
189	2454	
190	2467	
191	2480	
192	2493	
193	2506	
194	2519	
195	2532	
196	2545	
197	2558	
198	2571	
199	2584	
200	2597	
	Subtotal on this page :30306	Footer Row
	Subtotal :260700	
		Field

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 occurrence 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.

7.13.3 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 separated 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:

Header	123
This is the header text which will be displayed as page header	
Footer	123
This is the footer text - displayed in the footer area	
Data	
Text before the group	
Group "LOOP10"	
Group "DATA"	
Header	123
Header Row	
Data	
Data Row	
Footer	
Footer	
Footer Row	
Group "DATA"	
.... now start again	
Group "LOOP10"	
Text after the group.	

(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.

Note: Using the property `ColumnWidthSnapValue` You can control that the column widths which are very close (max. difference `ColumnWidthSnapValue` twips) are set to the same value. The default value 15 makes sure that one screen pixel difference does not count. This mode can be deactivated in the property "Options".

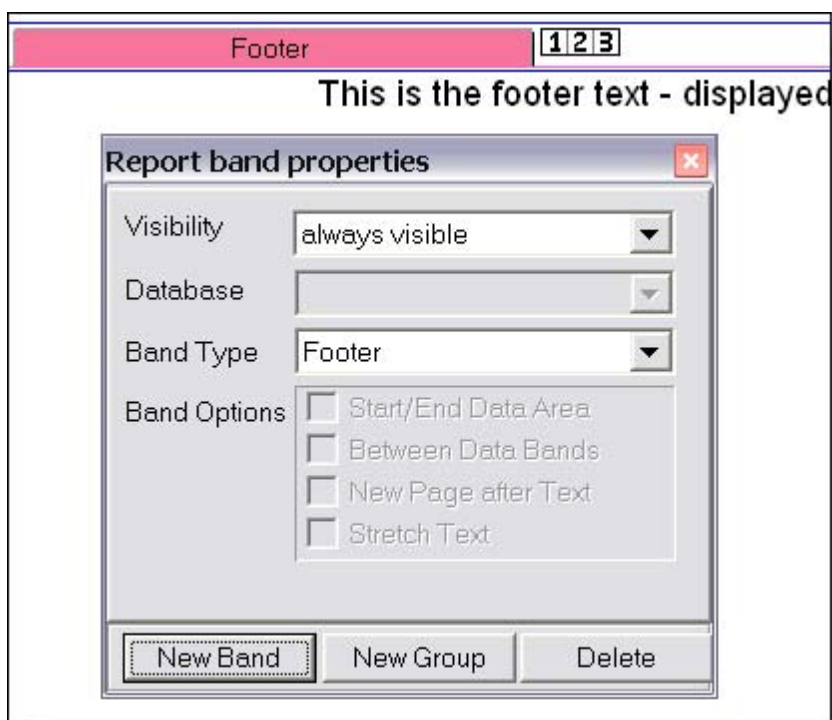
If you have long field names and want to **avoid word wrap** in your template, You can either apply the property `WPAT_NoWrap` to the cells which should not wrap or you can shorten the fieldnames automatically using `MergeText`, event `OnMailMergeGetText`. You can still show the complete fieldname in an hover event - please see chapter MailMerge. Note that `WPAT_NoWordWrap` must be enabled in property `RichText.FormatOptionsEx`.

Please note that the word wrap is highly influenced by the way the font width is calculated by the screen driver.

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 you 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.

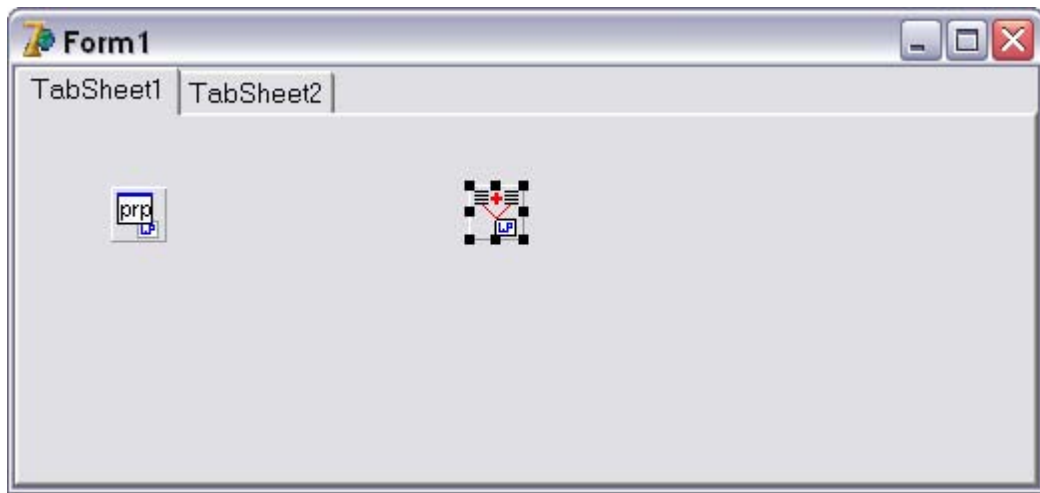
After one paragraph of the template has been copied to the destination control the fields in this paragraphs are replaced by data values. This work is performed by the mail merge function which can be also used without WPreporter. The mailmerge function triggers the OnMaileMergeGetText event for each 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.

7.13.4 WPreporter - step by step

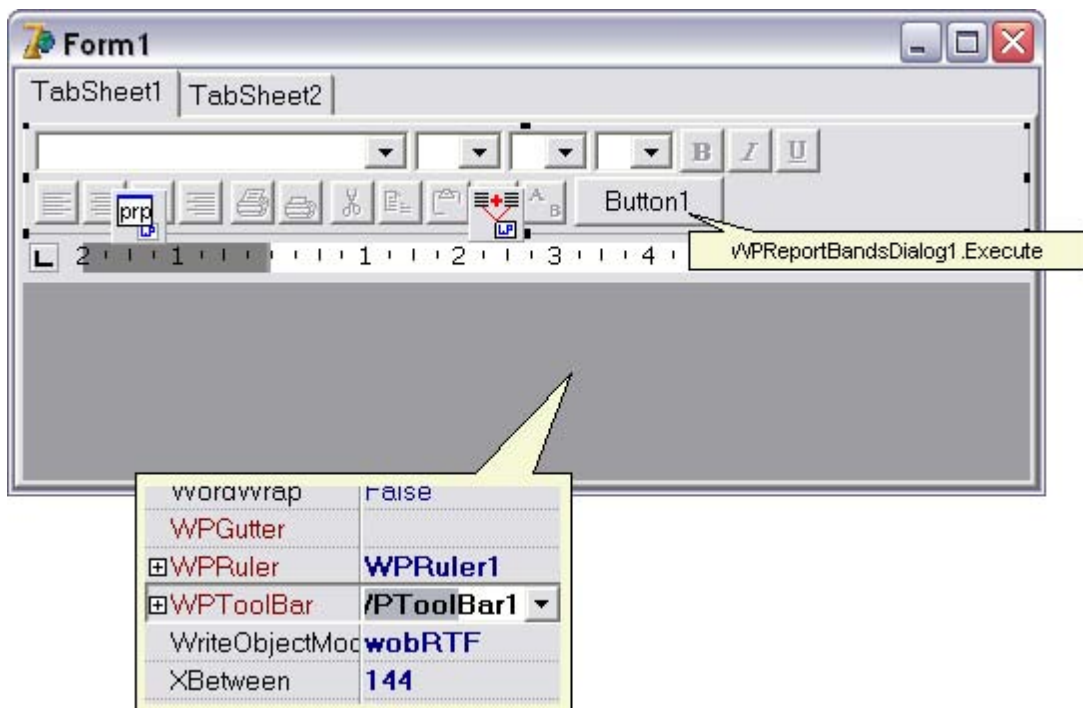
In this chapter we show how to build an application which creates a certain report.
(full source in directory "Tasks\WPreporter_MasterClient")

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 WPRReportBandsDialog1 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.

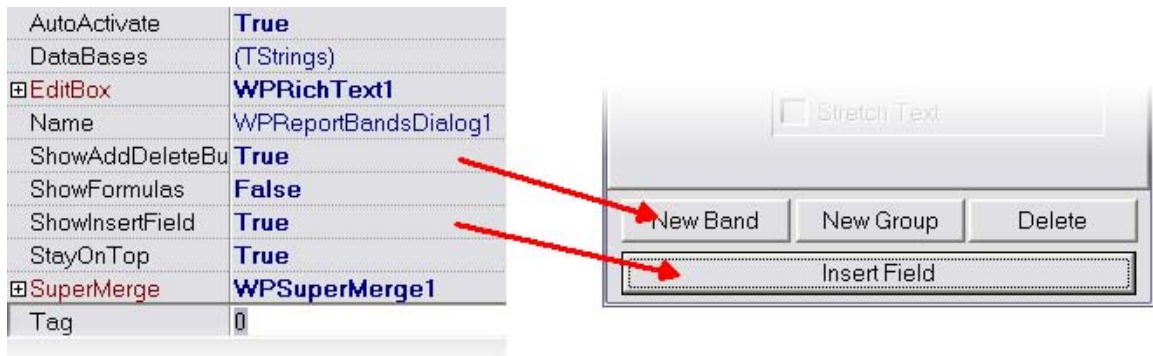
```

procedure TForm1.FormCreate(Sender: TObject);
begin
    WPSuperMerge1.SetSourceDest(
        WPRichText1.Memo.RTFData,
        WPRichText2.Memo.RTFData );

```

```
WReportBandsDialog1.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 relationship 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. The field names can optionally use the syntax **name=label_text**. If the equation sign is used the part after it will be displayed in the insert-field menu of the report editor dialog.

```
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
        + '=' + DataModule1.CUSTOMER.Fields[i].DisplayLabel );
    end;
  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
            + '=' + DataModule1.CUSTOMER.Fields[i].DisplayLabel );
    end;
end;

```

7) Now we want to implement a procedure which creates a generic template which works with our data.

Was add a button:  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 field names but can also collect the fields from the field names in the collection Field which we initialized in step (6).

Similar to 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:

Group "CUSTOMERS"												
Data												
«CustNo»	«Company»	«Addr1»	«Addr2»	«City»	«State»	«Zip»						
«Country»	«Phone»	«FAX»	«TaxRate»	«Contact»	«LastInvoiceDate»							

Group "ORDERS"												
Header												
Orders												
Data												
«OrderNo»	«CustNo»	«SaleDate»	«ShipDate»	«EmpNo»	«ShipToContact»	«ShipToAddr1»						
«ShipToAddr2»	«ShipToCity»	«ShipToState»	«ShipToZip»	«ShipToCountry»	«ShipToPhone»							
Footer												

wpCreateSmallHeaderRow

wpCreateSmallFooterRow

merged with: if ColNr>1 then include(par.prop, paprColMerge)

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 OnMailMergeGetText. In this event we locate the field in the specified 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: TWPMInsertTextContents);
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;

```

It is also possible to insert formatted text using the event OnMailMergeGetText. For example if you have a letter stored in a database or a different editor. Simply assign RTF, WPTOOLS or HTML code to Contents.StringValue.

If the inserted text starts with a table we recommend to set the mail merge option **mmDeleteThisField**. With this option the fields are deleted in the destination editor. So no paragraph has to be created above a table to store the field markers.

```

procedure TWPrepForm.WPSuperMerge1MailMergeGetText(Sender: TObject;
  const inspname: string; Contents: TWPMInsertTextContents);
begin
  if CompareText(inspname, 'RTFTEXT') = 0 then
    begin
      Contents.StringValue := ARTFTEXT.AsANSIString('RTF');
      Contents.Options := Contents.Options + [mmDeleteThisField];
    end else ....
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.

Header

123

Company Report

Footer

123

Page of 1 of 1

▼ Group "CUSTOMERS"

Data

«CUSTOMERS.Company»				«Addr1»		«Addr2»
«CustNo»	«Zip»	«Country»	«City»	«FAX»	«Phone»	«Contact»

▼ Group "ORDERS"

Header

Orders

Data

«ORDERS.OrderNo»	«ORDERS.CustNo»	«ORDERS.SaleDate»		«ORDERS.ShipDate»	«ORDERS.EmpNo»
------------------	-----------------	-------------------	--	-------------------	----------------

Footer

▲ Group "ORDERS"

▲ Group "CUSTOMERS"

The first, new header and footer bands create page header and footer. The page numbers have been inserted with `WPRichText1.InputTextField(wpoPageNumber)` - this function can be also selected from the InsertField dropdown in the report editor dialog.

This is the created report:

Company Report						
Kauai Dive Shoppe						
1221	94766-1234	US	Kapaa Kauai	4-976 Sugarloaf Hwy	808-555-0269	Suite 103 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						
1231		Bahamas	Freeport	PO Box Z-547 809-555-4958	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						
1351		Cyprus	Kato Paphos	1 Neptune Lane 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

Advanced Techniques:

Use the BeforeFormatTable event to force certain tables to be on one page.

Using the FormatOptions you can force all tables to be left intact (unless they are too long for a

page). Using the RTFDataCollection event BeforeFormatTable you can enable this mode for certain tables:

```
WPRichText2.HeaderFooter.BeforeFormatTable := BeforeFormatTable;  
  
procedure TWPrepTest.BeforeFormatTable(RTFData : TWPRTFDataCollection;  
    tablepar: TParagraph; var KeepTogether : Boolean);  
begin  
    if tablepar.RowCount=2 then  
        KeepTogether := TRUE;  
end;
```

7.13.5 WPreporter Events

Reporting with WPreporter is controlled by events. This makes it possible to work with any database system, or also without a database, for example if load calculated data.

Within the event code you can check the properties of the object WPSuperMerge.**Stack** to get information about the current position within the report template. The property Stack.Previous can be used to check the properties of the parent group of bands. Other useful properties are Stack.GroupBand, Stack.Band and Stack.CurrentParagraph.

This events are published by the class TWPSuperMerge:

OnMailMergeGetText - the most important event, it is used to insert the data for certain data fields.

This event uses the same parameters like the TWPRichText event OnMailMergeGetText. In the simplest case the event handler can be

```
procedure TForm1.DoMailMergGetTexte(
  Sender: TObject; const inspname: string;
  Contents: TWPMInsertTextContents);
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.FieldName(inspname).AsString;
```

You can also change the attribute of the inserted text using ['MergeAttr'](#).

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

BeforeProcessGroup:

This event is the second most important. It is used to control if a group should be processed or not. If the group should be processed set the parameter `ProcessGroup` to true, if it should be not processed set the parameter `ProcessGroup` to false. If you know that this would be the last time this group is being used set the parameter `IsLastRun` to true.

When the template uses nested groups the `BeforeProcessGroup` will be triggered for the nested group before `AfterProcessGroup` is triggered for the outer group.

Typically you test the `DataSet` for being at the end (EOF) and set the parameter `ProcessGroup` accordingly:

```
if Band.Alias = 'MAIN' then ProcessGroup := not MainDataSet.EOF;
```

OnPrepareHeader:

This event is triggered before a header band is being processed. You may set the parameter `Abort` to true to skip the band.

OnPrepareText:

This event is triggered before a text band is being processed. You may set the parameter `Abort` to true to skip the band.

OnPrepareFooter:

This event is triggered before a footer band is being processed. You may set the parameter `Abort` to true to skip the band.

OnPostProcessBandData:

This event makes it possible to modify a paragraph after it was created in the destination text buffer. It is triggered after the entire text for a band was created. The start and end paragraphs are passed to the event. The paragraph references can refer to normal paragraphs or table row paragraphs. You can use this event to add flags to paragraphs to mark certain areas in the report.

AfterProcessGroupData:

This event is processed after the data bands in a group have been processed. You can use

it to sum up values which should be used as totals in footers. AfterProcessGroupData will be triggered each round the group is used. You can set WPSuperMerge.Stack. PagebreakAtGroupEnd = true inside this event to force a pagebreak after the group data.

AfterProcessGroup:

This event is triggered after a group has been processed. Typically you use this event to advance to the next record in the database.

```
if Band.Alias = 'MAIN' then MainDataSet.Next;
```

7.13.6 Convert text into template

One unique strength of our WPSuperMerge concept is the possibility to convert an existing report (maybe you used to create it using OLE before) quickly into a report template.

You can use the procedure **ConvertLetterIntoTemplate**. This procedure creates a report with header and footer bands which are initialized to create the header and footer texts found in the original document. Around the body of the document a group is created which makes it easy to create many letters in a row.

Also see:

[Token to Template Conversion](#)

[General Syntax - Fields](#)

[Syntax Highlighting](#)

[Bands](#)

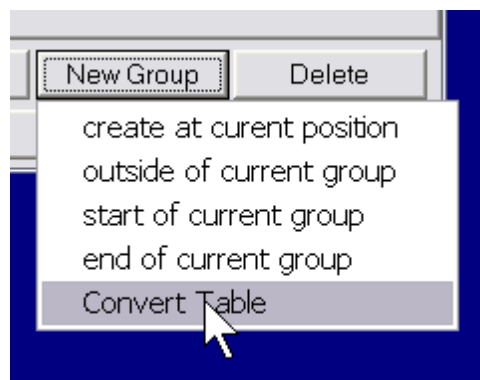
[Groups](#)

[Parameters for Fields](#)

[Parameters for Bands and Groups](#)

[Example Template](#)

- 1) All texts which are fixed have to remain in the template
- 2) All text blocks which depend on selected data must be replaced by mail merge fields.
- 3) Tables can be converted into groups with the utility function SuperMerge.
ConvertTableIntoGroup. This utility can be called from within the report editor dialog. The tool can be used when the cursor is inside of a table, please do not select the table.



existing table in document

Unit Sales	Jan06	Feb06	Mar06	
Retail	60	60	60	
Fleet				
Total	60	60	60	
New Vehicle Net	\$ 56	\$ 56	\$ 56	

→ template group

Group					
Header					
Unit Sales	Jan06	Feb06	Mar06	Apr06	May06
Data					
Retail	60	60	60	60	60
Footer					
New Vehicle Net	\$ 56	\$ 56	\$ 56	\$ 56	\$ 56
Group					

Sometimes it is a good idea to simplify the table before you use "ConvertTableIntoGroup". Very useful will be the API **'SplitTable'** to split up a table which contains data from different logical elements.

4) Make text blocks conditional:

Please add a possibility to the application to set the Name of a band. i.e. you can use a combo box or popup menu.

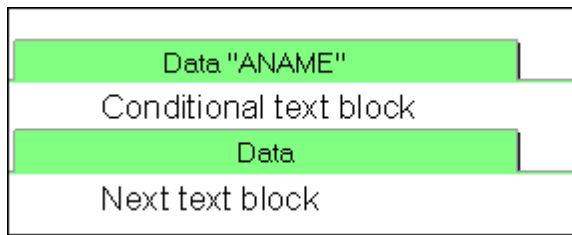
The code would be basically:

```
if WPSuperMerge1.Bands.CurrentBand<>nil then
    WPSuperMerge1.Bands.CurrentBand.Alias := 'ANAME';
```

With the **OnPrepareText** event You can then check text bands (the green ones). If the band uses a certain Alias (alternatively You can use property Name) You can set the variable 'Abort' depending on a certain condition.

```
procedure TWPrepTest.WPSuperMerge1PrepareText(Sender: TWPSuperMerge;
    Band: TWPBand; Count: Integer; var ParProps: TParagraphProperty;
    var Abort: Boolean);
begin
    if Band.Alias = 'ANAME' then
    begin
        Abort := TRUE;
    end;
end;
```

Now you can insert a data band and assign the 'Alias':



Note: If you assign a string to property 'Name', this text will be displayed instead of 'Data'

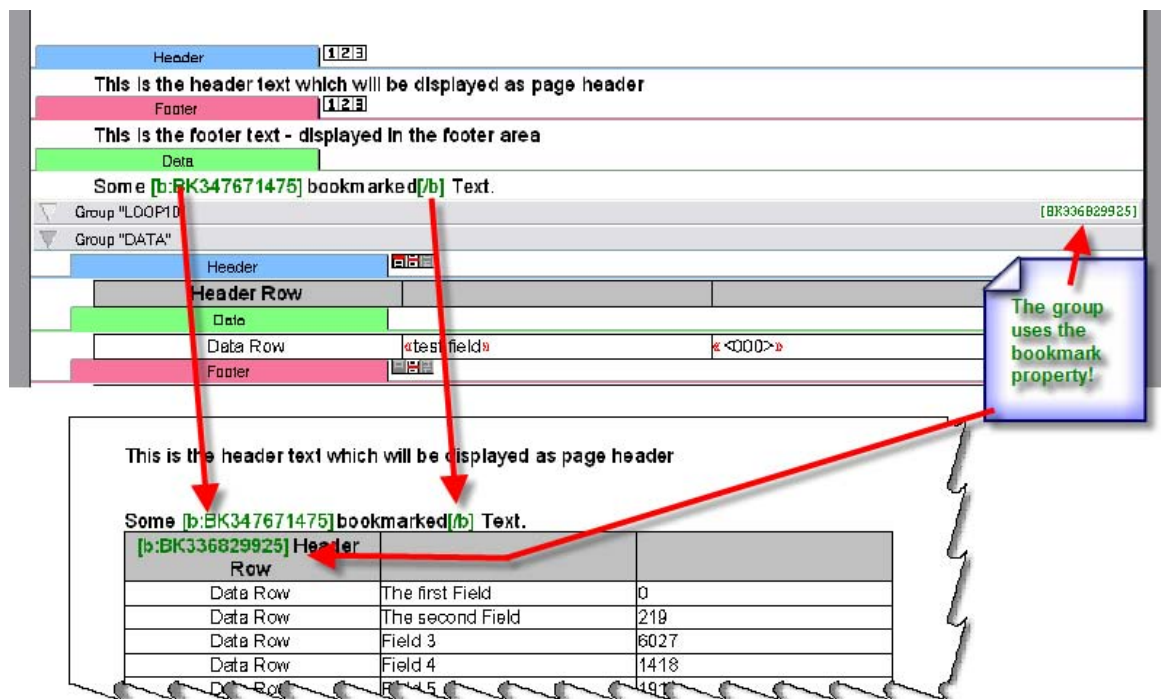
7.13.7 Reporter and Bookmarks

WPSuperMerge is able to create bookmarks around the text which was created in the destination document for a certain band or group.

This bookmarks will be created after the text was entirely created if the property 'Bookmark' was set to a non empty string.

Bookmarks (TWPTTextObj objects) embedded in the template will also be copied to the destination, but since it is not possible to place bookmarks in band paragraphs the above mentioned functionality is the only way to do it.

In this example template embedded bookmarks are used. The first group uses the bookmark property.



Note: The bookmarks are usually invisible. They are displayed only if the flag **wpShowBookmarkCodes** was used in property **FormatOptions**.

7.13.8 Token to Template Conversion

If you use the *token to template conversion* you can edit the reporting template with an editor such as MS Word. This feature has been added to WPTools 7 with the WPReporter addon.

The template is loaded into TWPRichText. In this control the template can be further edited with the additional convenience of syntax highlighting and on demand converted into a "true" reporting template. (The latter uses nested paragraphs to represent groups and special paragraph types for bands. Fields are not text based but use objects instead.).

Tip: The report templates can be used further in our .NET and OCX components TextDynamic and TextDynamic Server.

If you used the mail merge feature before and have documents with embedded merge fields those can be easily used a reporting templates as well! It is also possible to convert an existing report into a template.

To activate syntax highlighting only one line of code is required:

```
SourceText.CustomSyntax := TWPFielfBandSyntax.Create(nil);
```

The created instance of [TWPFielfBandSyntax](#) will be freed automatically.

To convert the text to a reporting template simply execute the method **WPConvertReportScript from unit WPRTReport.**

```
function WPConvertReportScript(
  Source: TWPRTFDataCollection;
  Mode: TWPCConvertReportScriptMode
    = [wpClearErrorsAndWarning,
      wpConvertFields,
      wpConvertBands,
      wpConvertGroups];
  // If this is nil the 'Body' will be converted
  DataBlock: TWPRTFDataBlock = nil;
  // Parameters used to detect fields and bands
  FieldStart: WideString = '<<';
  FieldEnd: WideString = '>>';
  BandChar: WideChar = ':';
  GroupChar: WideChar = '#'): TWPCConvertReportScriptResult;
```

Example:

```
procedure TWPrepForm.ConvertClick(Sender: TObject);
begin
  WPConvertReportScript( SourceText.RTFData );
  SourceText.ReformatAll(true,true);
end;
```

In case You don't want to destroy the original text with the tokens You can also use a second editor "TemplateText" and this code:

```
TemplateText.Assign(SourceText);
```

```
WPConvertReportScript( TemplateText.RTFData );  
TemplateText.ReformatAll(true, true);
```

The report is then created in the output editor "ReportText" by:

```
ReportText.Clear;  
WPSuperMerge1.Execute;
```

It is important that the super merge component has been correctly set up

```
WPSuperMerge1.SetSourceDest(TemplateText.RTFData, ReportText.RTFData);
```

7.13.8.1 General Syntax - Fields

The token to template conversion uses special character combinations to separate the commands from the text. Fields have to be embedded into the characters << and >>.

(This codes <<, >>, : and # can be customized in [TWPFfieldBandSyntax](#) for syntax highlighting. Other values can be passed to the method WPConvertReportScript)

Note:

The parser expects that after the << and before the >> characters no whitespace characters are typed. Otherwise the characters will be interpreted as text.

Example for fields:

<<name>>

The name of a field may contain space characters.

Optionally the closing '/' character can be printed:

<<name/ >>

Using such fields mail merge documents can be created. Such merge documents only require fields to be filled with data. Using mail merge does not require the reporting extension so we decided to make the syntax highlighting and token to template conversion available in the basis edition.

If you need repeated data rows in a document you need groups and bands. Bands mark certain text to be the header and footer of a document or group while groups are used to loop certain parts of the template as long as data to fill in the fields is available. Groups can also be used to disable certain parts of the template. While bands are not nested, groups can be nested and so consist of a start and end token.

Note: The name of a field will be passed to the event [WPRichText.OnMailMergeGetText](#) or [WPSuperMerge.OnMailMergeGetText](#) while merging the text or creating a report. This event can retrieve the text which should be inserted inside the field.

7.13.8.2 Syntax Highlighting

WPTools includes a **syntax highlighter** for XML and also for the syntax described in this chapter (see [TWPFeldBandSyntax](#))

Using the syntax highlighting for report templates is easy - and - the highlighting works non destructive!

Show highlighting:

```
SourceText.CustomSyntax := TWPFeldBandSyntax.Create(nil);
```

```
<<:HEADERF "my header"/>> -- any text after the group will be
    ignored as comment
This is the header on the first page
<<:HEADER "my header"/>> -- HEADERF, HEADERO, HEADERE, HEADER
This is the header on all pages
<<:TEXT>> -- start the regular text
```

Dear Developer,

Remove highlighting:

```
SourceText.CustomSyntax := nil
```

```
<<:HEADERF "my header"/>> -- any text after the group will be ignored as comment
This is the header on the first page
<<:HEADER "my header"/>> -- HEADERF, HEADERO, HEADERE, HEADER
This is the header on all pages
<<:TEXT>> -- start the regular text
```

Dear Developer,

Please note, how the original text attributes are restored - here the bold characters in the third line.

7.13.8.3 Bands

Bands are identified by a colon (':') after the opening << characters.

Example for header bands:

```
<<:HEADER>>
```

This starts a group header or, if used outside of any group a document header text. In the latter case the following variations are possible:

```
HEADERF = header on first page only
HEADERO = header on odd pages
HEADERE = header on even pages
HEADER_OFF = this header is disabled.
```

Please note that bands are not implemented using opening and closing tokens ("i.e. <<...>>...<</...>>") so optionally the closing '/' character can be typed: <<:HEADER/>>

Example for footer bands:

```
<<:FOOTER>>
```

This starts a group footer or, if used outside of any group a document footer text. In the latter case the following variations are possible:

FOOTERF = footer on first page only
 FOOTERO = footer on odd pages
 FOOTERE = footer on even pages
 FOOTER_OFF = this footer is disabled.

Inside header and footer bands fields, images and text is possible. A header and footer band ends where either a different header or footer band starts or a data band:

<<:DATA/>>

Bands must be the **first non white space** (white space= space or tab characters) in a paragraph. All text after the band token will be ignored and can be used to write comments.

After the name, i.e. HEADERF You can specify an alias value: **<<:HEADER "the alias">>**

This Alias will be passed to the band event of WPSuperMerge component.

Hint: It is also possible to change the names. They are defined in unit WPRTEReport in this array variables:

```
WPHeaderNames: array[TWPMergeShowOptions] of string =
  ('HEADER', 'HEADERO', 'HEADERE', 'HEADERF', 'HEADERNF', 'HEADERONF', 'HEADER_OFF');
WPFooterNames: array[TWPMergeShowOptions] of string =
  ('FOOTER', 'FOOTERO', 'FOOTERE', 'FOOTERF', 'FOOTERNF', 'FOOTERONF', 'FOOTER_OFF');
WPDATANames: array[1..2] of string = ('DATA', 'TEXT');
```

You can access the alias name, i.e. "the alis" in band.Alias

7.13.8.4 Groups

Groups are identified by a double cross ('#') after the opening << characters. While bands <<:.../>> are not nested, groups can be nested and so are embedded into a tag pairs **<<#...>> <<#/....>>**.

Unlike header and footer bands the name of the group tags are not fixed. Practically any name can be used if it does not contain spaces. However the opening token must match the closing token. For practical reasons the names should match the database which is referenced inside of the group.

Example:

```
<<#CUSTOMERS>> comment: we list all customers in this group
  <<Customers.Name/>>
  <<Customers.Address/>>
  Ordered Items:
  <<#ORDERS>> comment: we list all items ordered by the current
customer
    <<Orders.Name/>>
    <<#/ORDERS>>
  <<#/CUSTOMERS>>
```

Note: The name of a group will be passed to the [band and group events](#) while creating

a report.

This event has to decide if a group should be processed (again) or not. It can create a sql query and calculate sub totals.

You can access the name, i.e. "CUSTOMERS" in band.Name.

7.13.8.5 Parameters for Fields

1. Name - may contain spaces! (required)

If first character is the @ sign, the complete text will be handled as formula. (no other options are possible)

2. Displayname in " " - must be second position !

3. Options.

- +spc - this will add a space after the field if it was NOT empty.
- +spc/ - this will add a space before the field if it was NOT empty
- +nl - to add a new-line if it was not empty
- \f "some text" - will be added after the field if it was NOT empty. Alias: + "..."
- \b "some text" - will be added before the field if it was NOT empty. Alias: + ".../"
- "some text" - will be printed if it WAS empty
- \r "some text" - will be printed instead of the field

Special attributes

- \remove - the field will be removed
- \setattr - the paragraph attribute will be overwritten by field contents attributes
- \autosetattr - if this is first field in line use the para attribute
- \keepattr - protect the current paragraph attributes
- \loadimage - interpret field value as file name of an image file
- \loadtext - interpret field value as file name of a text file

Supports Word Syntax: \b before \f after. Spaces are allowed after +, -, \b, \f ...

4. Format @"...."

\@ "%f" - uses this formatting code to format the field (usually floating point numbers)

5. Condition ?...=...

- ?fieldname=null - use this field if the specified field was null
- ?fieldname#null - use this field if the specified field is not null
- ?fieldname=zero - use this field if the specified field was zero
- ?fieldname#zero - use this field if the specified field is not zero

Modify: [Contents.IsNull](#) to tell the engine that a field is null or zero.

7.13.8.6 Parameters for Bands and Groups

1. Name - predefined header/footer names or one of the possible group names
2. Displayname name in " " - must be second position

3. Option +/-

- +ff - start a new page after the band/group
- ff - start a new page before the band/group

Only header and footer: ("<<:HEADER>>")

- +afterstretch - header/footer used after a stretched band
- +between - use band between records
- start - don't use header at start
- end - don't use footer at end

Only Data Bands ("<<:DATA>>")

- stretch - deactivate stretching (default = on)
- +keep - keep this paragraph together with next
- +newtable - start a new table

4. Condition ?...=...

The following conditions to use a band are possible:

- =null - field must be null
- #null - field must not be null
- =zero - field must be zero
- #zero - field must not be zero
- ="..." - the field must have a certain value
- #"..." - the field must be different to a certain value

Example:

```
<<:data ?gender="W">>¶
Dear Mrs. <<name/>>,¶
<<:data ?gender="M">>¶
Dear Mr. <<name/>>,¶
```

Modify: [Contents.IsNull](#) to tell the engine that a field is null or zero.

5. Formula - must be last option

@....

It is used as start condition in a group start and a continue condition if used in a group end token.

Any text in the paragraph after the closing '>>' will be ignored as comment!

7.13.8.7 Example Template

```
<<:HEADERF "my header"/>> -- any text after the group will be ignored as
comment
This is the header on the first page
<<:HEADER "my header"/>> -- HEADERF, HEADERO, HEADERE, HEADER
This is the header on all pages
<<:TEXT>> -- start the regular text

Dear Developer,

This letter documents and demonstrates the reporting feature which is based on a
```

template written in regular formatted text with additional tags:

The "<<#" signs are expected as first non-space signs in a paragraph. Any text after the closing >> signs is ignored and can be used as comment. Fields are inserted between the signs << and >>. It is important that after the '<<' no white space or punctuation is written.

Headers in groups are written using single, not open/closing tags. They cannot be nested anyway. The / before the closing >> signs should be added but is not required.

Only groups are nestable. Groups are started using the code <<#. The name after the # sign will be the group name, so a database name can be used. An optional alias can be specified. It is possible to set up a list of possible group names.

Now our list starts:

<<#GROUP1 ?name#null "Invoice">> -- the table will be given the display name "invoice", the logical names remains "GROUP1". After the optional name a condition is expected.

<<:HEADER/>> -- this header will be displayed at the start of the group Group Header. Fields are possible: <<fieldname +spc/>>

<<:DATA ?"has orders"=null/>> -- this is the data row.

Some data here - under condition that there are no orders. (Note that a field name may used spaces. In this case use " in the field name part as well.)

<<:DATA ?hasorders#null/>> -- this is the data row.

Some data here - under condition there are orders.

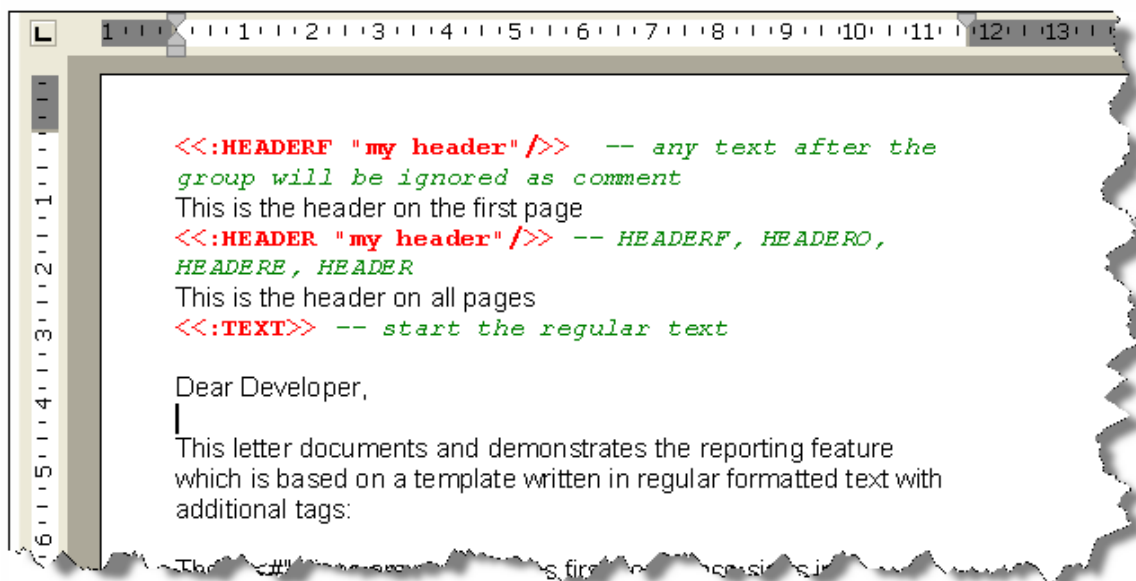
<<:FOOTER/>> -- this footer will be displayed at the end of the group

Group Footer. Fields are possible: <<fieldname/>>

<<#/GROUP1>> -- close the group.

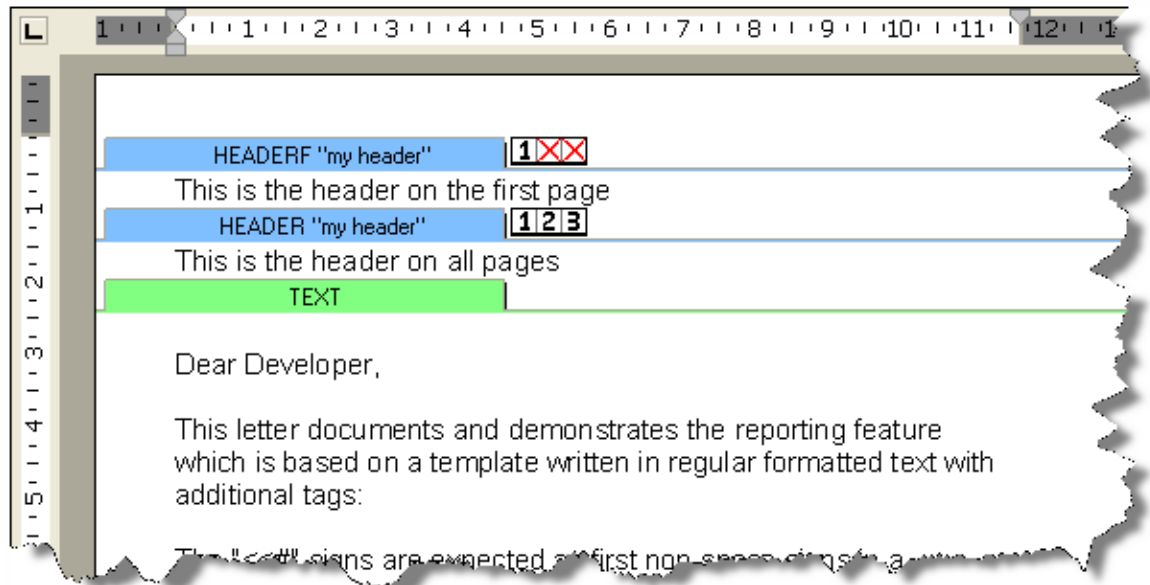
This text comes after the group.

When viewed with activated the [syntax highlighting](#) the document will look like



Note: The syntax highlighter does not make any modifications to the attributes of the text. All highlighting is done just visual and is updated while the text is edited.

After the conversion to the internal reporting template structure the document looks like this:



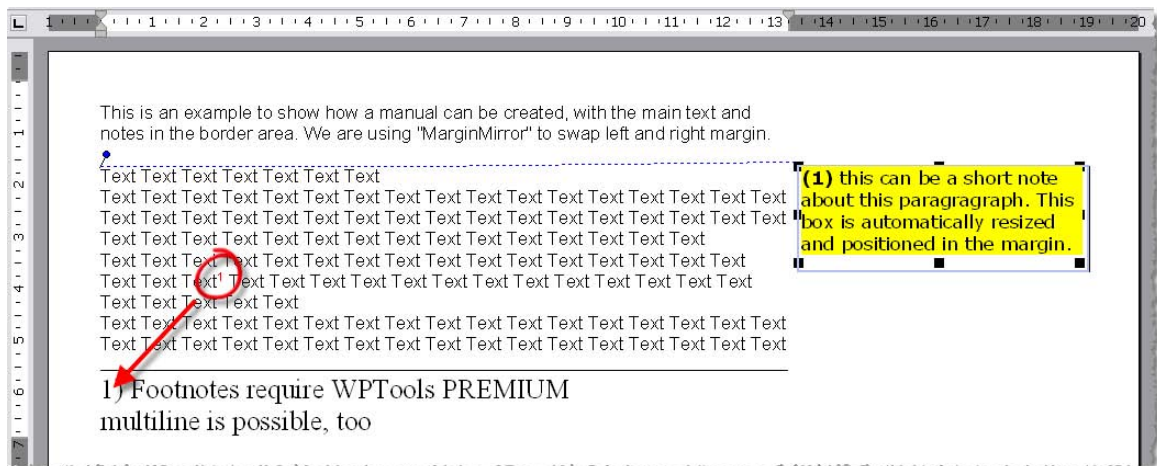
7.14 L) WPPremium Addon

WPTools Premium adds the support for footnotes, text boxes and (soon) columns to the edition WPTools Professional Bundle. It comes with 100% Source code.

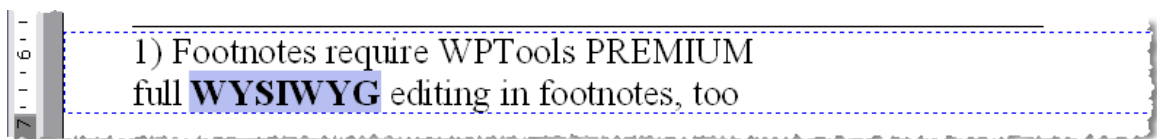
If you have purchased WPTools **Premium** please see the dedicated WPTools Premium manual (PDF) which also includes interesting technical information.

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:

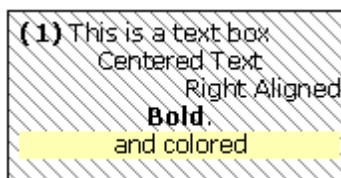


7.14.1 Text boxes

Text boxes are implemented as a special TWPObj class, TWPORTFTextBox from unit WPOBJ_TextBox. As mentioned in the manual the TWPObj instances are referenced by an instance of the TWPTxtObj class which is hosted by the TParagraph which owns it. The TWPORTFTextBox does not store the text, that is a data block in the data collection which owns the object and the host paragraph.

So actually it would have been possible to paint the boxes without implementing the TWPORTFTextBox class too. But the TWPObj concept is very useful since it offers a number of important public and internal procedures, last but not least the procedure WriteRTFData which is used when the text is written. This concept makes sure the logic is abstracted enough to make modifications easy. So you can implement modifications in the quite easy to understand unit WPOBJ_TextBox instead to deal with the RTF reader or writer or even the RTF engine with its thousands of code lines. You can also modify the paint routine to paint a background pattern.

Example: to make a text box appears with a background:



use the code

```
procedure TWPORTFTextBox.Paint(toCanvas: TCanvas; BoundsRect: TRect;
  ParentTxtObj: TWPTxtObj; PaintMode: TWPTxtObjectPaintModes);
begin
  if toCanvas <> nil then
  begin
    toCanvas.Pen.Style := psSolid;
    toCanvas.Pen.Color := clBlack;
```

```

        toCanvas.Pen.Width := 0;
        toCanvas.Brush.Color := clGray;
        toCanvas.Brush.Style := bsFDiagonal;
        toCanvas.Rectangle( BoundsRect.Left, BoundsRect.Top, BoundsRect.Right,
        BoundsRect.Bottom );
    end else inherited Paint(toCanvas, BoundsRect, ParentTxtObj, PaintMode);
end;

```

But how does the engine knows which text block belongs to which text box?

For this connection the property ObjName is used. The value of this property is used by function GetRTFDataBlock to locate the correct RTF Data block. (Of course, other solutions are thinkable)

```

function TWPORTFTextBox.GetRTFDataBlock : TWPRTFDataBlock;
begin
    if (FDataCollection<>nil) and (ObjName<>'') then
        Result := FDataCollection.Find(wpIsOwnerSelected, wpraNamed, ObjName)
    else Result := nil;
end;

```

Here you see that the 'Kind' of the RTFDataBlock which is used for text boxes is wpIsOwnerSelected, the 'Range' is wpraNamed and of course it has a name.

From the code above you also can see that the RTFDataBlock is not automatically created when the function GetRTFDataBlock is used. It must be created by code such as FDataCollection.Get (wpIsOwnerSelected, wpraNamed, ObjName). This is automatically done by procedure TWPORTFTextBox.SetAsString.

Note: Do not expect the names to be the same after a file was saved and loaded in RTF format. Only the WPTools format will preserve the object names.

The following code inserts a text box by creating an instance of TWPORTFTextBox. (Tip: You can of adapt htis code to use the TParagraph.InsertObject procedure if you do not want to insert at cursor position.)

```

var obj: TWPORTFTextBox;
    txtobj: TWPTTextObj;
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):



Alternatively you can also use `WPRichText1.TextObjects.InsertTextBox`:

```

procedure TForm1.WPRichText1DragDrop(Sender, Source: TObject; X,
  Y: Integer);
var
  aField : String;
  w, h: Integer;
  txtobj: TWPTextObj;
  RTFDataBlock : TWPRTFDataBlock;
  PageNr, PageXTW, PageYTW : Integer;
  par : TParagraph;
begin
  If (Source=FieldList) And (FieldList.ItemIndex>=0) And
    WPRichText1.GetPageXYfromXY(X,Y,PageNr, PageXTW, PageYTW,par) Then
    begin
      aField := FieldList.Items[FieldList.ItemIndex];
      w := WPCentimeterToTwips(5);
      h := WPCentimeterToTwips(0.5);
      // Insert the box
      WPRichText1.TextCursor.MoveTo(par,0,False);
      txtobj := WPRichText1.TextObjects.InsertTextBox(w,h,RTFDataBlock);
      // Set contents
      RTFDataBlock.RtfText.AsString := aField; // HTML would be possible Or RTF !
      // Now update the txtobj
      txtobj.Source := aField;
      txtobj.PositionMode := wpotPage;
      txtobj.Wrap := wpwrNone;// Important For page objects
      txtobj.RelX := PageXTW; // X In twips !
      txtobj.RelY := PageYTW; // Y In twips !
      // We want To resize it freely
      // disable the autosizing (but at least the contents must fit!)
      txtobj.Mode := txtobj.Mode + [wpobjDisableAutoSize];
    End;
  End;

```

This code is used in the demo BoxAtPos which shows how to use textboxes to build a form (using absolute positions!)

Tip: To create a text box which is automatically positioned in the margin of the page use the modes `wpobjLockedPos`, `wpobjPositionInMargin`

Tip: You can assign a name to a text box. Use the property `TWPTextObj.Source` (not property `Name`!). You can move the cursor to this text box using the code

```

WPRichText1.CodeMoveTo('name', wpobjImage, [], [wpCompareSource]);
WPRichText1.SetFocus;

```

To automatically enter the text box use

```

if WPRichText1.CodeMoveTo('name', wpobjImage, [], [wpCompareSource]) and

```

```
WPRichText1.CPObj.IsImage
then WPRichText1.CPObj.ObjRef.Edit;
```

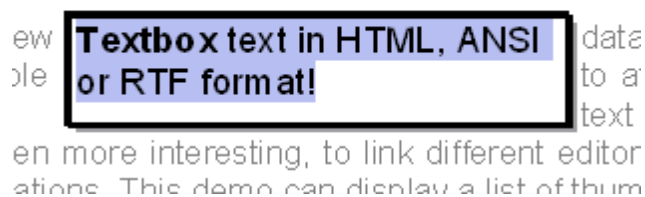
To insert a text box use this code:

```
var obj: TWPORTFTextBox;
    txtobj: TWTextObj;
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;
end;
```

This code in the **OnTextObjectDbClick** event makes the text box editable:

```
if pobj.ObjType = wpobjImage then
begin
    if pobj.ObjRef <> nil then pobj.ObjRef.Edit;
    ignore := TRUE;
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`

7.14.2 Footnotes

Footnotes are implemented similar to text boxes. Like text boxes the contents are stored in the RTFDataCollection. But footnotes do not use a TWPObj class to be painted, they use just one TWPTextObj instance with ObjType = wpobjFootnote.

Text text of the footnote uses the style mentioned in the StyleName property of the TWPTextObj as basis style.

So a footnote can be created with the following code. This code also check if a paragraph style names 'Footnotes' exists and, if not, creates one.

The code also create a default text with an text object named 'TEXTNR'. This text object will display the number of the footnote.

At the end the complete text of the footnote is selected, except for the number.

```

procedure TWPTextBoxDemo.InsertFootnoteClick(Sender: TObject);
var foottext : TWPRTFDataBlock;
    footobj : TWPTextObj;
begin
    // Create default footnote style:
    if WPRichText1.ParStyles.GetID('Footnotes')=0 then
    begin
        with WPRichText1.ParStyles.AddStyle('Footnotes') do
        begin
            ASetFontName('Times New Roman');
            ASet(WPAT_CharFontSize, 600);
        end;
    end;

    inc(FootNoteNr);
    WPRichText1.CurrAttr.AddStyle([afsSuper]);
    footobj := WPRichText1.TextObjects.InsertNewObject(wpobjFootnote,false, false);
    WPRichText1.CurrAttr.DeleteStyle([afsSuper]);
    footobj.Name := 'FOOT_' + IntToStr(FootNoteNr);
    footobj.StyleName := 'Footnotes';
    foottext := WPRichText1.HeaderFooter.Get(wpIsFootnote, wpraOnAllPages, footobj.
Name);
    foottext.RtfText.AsString
        := '<html><textobj name="TEXTNR"/>' + Edit1.Text + '</html>';
    WPRichText1.CursorOnText := foottext;
    WPRichText1.CPPosition := 3;
    WPRichText1.SetSelPosLen(3, MaxInt);
    WPRichText1.ShowCursor;
    WPRichText1.SetFocus;
end;

```

Since this is bit much code to simple create a footnote we have implemented the function

InputFootnote(PlaceCursor: Boolean; CreateNumber: Boolean = TRUE; InitText: string = #32);
 which does basically the same as the above code.

Please use the method

InputFootnote(PlaceCursor: Boolean; CreateNumber: Boolean = TRUE; InitText: string = #32);

to create a footnote. Like textboxes, headers and footers the footnotes are also "text layers".

The text layers are all stored as collection items in the collection HeaderFooters.

7.14.3 Columns

WPTools Version 7 "Premium" supports columns.

Columns are controlled by the paragraph properties WPAT_COLUMNS and WPAT_COLUMNS_X. WPAT_COLUMNS specifies the number of columns and WPAT_COLUMNS_X the distance between.

Please note that "text balancing" is supported.

This code starts a 2 column layout at the current paragraph with 0.5 cm margin inbetween.

```
WPRichText1.ActiveParagraph.ASet(WPAT_COLUMNS, 2);  
WPRichText1.ActiveParagraph.ASet(WPAT_COLUMNS_X, WPCentimeterToTwips(0.5));  
WPRichText1.Refresh;
```

This code will create 3 columns

```
WPRichText1.ActiveParagraph.ASet(WPAT_COLUMNS, 3);  
WPRichText1.ActiveParagraph.ASet(WPAT_COLUMNS_X, WPCentimeterToTwips(0.5));  
WPRichText1.Refresh;
```

With this code a column break is inserted so this paragraph will be moved to next column or page. The user will need this functionality if a column should not be extended to the end of the page.

```
include(WPRichText1.ActiveParagraph.prop, paprNewColumn);  
WPRichText1.Refresh;
```

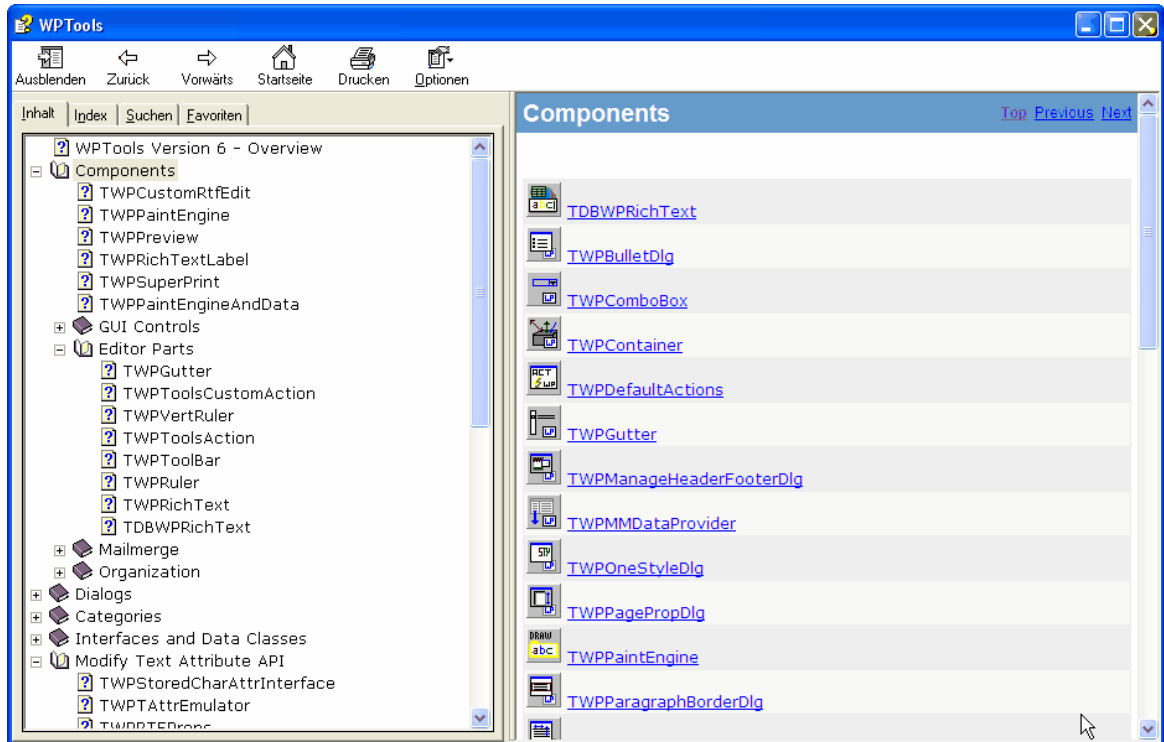
To switch of column layout use

```
WPRichText1.ActiveParagraph.ASet(WPAT_COLUMNS, 1);  
WPRichText1.ActiveParagraph.ADel(WPAT_COLUMNS_X);  
WPRichText1.Refresh;
```

Column properties are loaded and saved in WPT and RTF format.

8 Reference

The registered version of WPTools 7 also includes an organized CMH file which can serve as reference.

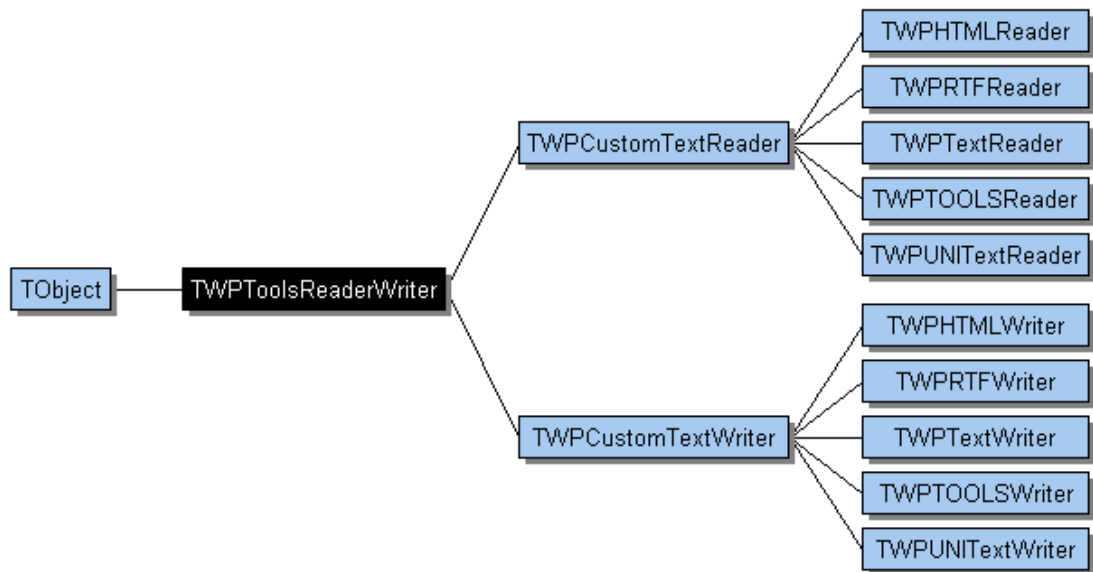


8.1 Reader and Writer

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.

It is possible to select a certain reader or writer using a 'format string'. This string is passed to load and save methods, such as `LoadFromFile` and also in the properties `TextLoadFormat` and `TextSaveFormat`.

To select a certain reader or writer simply name it in the **format string**. You can pass options to the reader or writer, too.

This options are appended to the format string after a minus sign, i.e "HTML-onlybody". Multiple options are separated by comma.

You can also set the code page for the loading and or saving operations. This can be useful with the data sensitive control `DBWPRichText`:

Example:

```
DBWPRichText1.TextLoadFormat := 'ANSI-codepage1251'; // cyrillic
DBWPRichText1.TextSaveFormat := 'ANSI-codepage1251';
```

In general these options are supported but not all reader/writer can support all options:

In our [web based forum](#) we have posted several FAQ topics and articles.

The article "[Compose an e-mail in HTML format / Save image to HTML](#)" shows how to create a HTML e-mail.

Overview

For details please see table here: <http://www.wpcubed.com/manuals/formatstrings.htm>

onlybody - only write body part
ignorefonts - do not load font information
ignorefontsize - do not load font size information
nostyles - do not load or save styles

nonumstyles - do not load or save number styles
noimages - do not load or save images
basertext - only save text in WPTOOLS format, no styles or other info
nomergefields - do not save merge fields, only contents
nohyperlinks - do not save hyperlinks
nobookmarks - don't save bookmarks
novariables - don't save or load RTF variables
nobinary - try to convert all binary to ASCII
nopageinfo - do not save or load page size information
ignorekeepn - don't load the keepn flag from RTF
ignorerowmerge - do not load row merging from RTF
ignorecollapsedpar - don't save table rows which are hidden by WReporter
codepage12xx - set the code page XY for loading. Default is 1252.
complete - load header and footer information although we are inserting text
alwaysembed - embed image data also for linked images

For further reference please see the implementation of the reader and writer classes in the units WPIO*.PAS. All options are processed by the procedure "SetOptions". The reader and writer publish "Opt_name" properties which are set to true when an format option with the same name was used.

8.2 TParagraph API

This object stores the text. It is organized in a linked list (NextPar/PrevPar) starting from TRTFData.FirstPar. Paragraphs can also contain sub-paragraphs (ChildPar).

Similar to WPTools Version 4 tables cells are handled as paragraphs. In contrast to previous versions, WPTools 7 handles child paragraphs to support multiple paragraphs in the one table cell. Even tables can be inside a table cell - (but they will always start under, and not beside the normal text since they always start on a new line.)

Here we list the most important properties and methods.

8.2.1 TParagraph Properties

ActCellWidth

AClass

align

ANSIChr

ANSIText

Cell

ChildPar

Children
ChildrenCount
ColCount
ColFirst
ColLast
ColLeft
ColNext
ColNr
ColPrior
ColRight
Cols
ColSpan
CustomParameters
EndSpanH
EndSpanV
FirstLine
FirstSibling
HasChildren
Hidden
id
indentfirst
indentleft
indentright
IsFirstTextPar
IsLastTextPar
IsNewPage
LastChild
LastInnerChild

LastLine
LastSibling
LineCount
LoadedCharAttr
LogNextPar
LogPrevPar
MinCHeight
next
NextPar
nextpardown
ObjectIndex
ObjectRef
ParagraphType
ParentCell
ParentGroup
ParentPar
ParentParentPar
ParentParentRow
ParentParentTable
ParentRow
ParentTable
ParLength
prev
PrevPar
RowCount
RowDown
RowFirst
RowLast

RowLogNr

RowNext

RowNr

RowPrior

Rows

RowSpan

RowUp

RTFData

Sibling

SiblingCount

SiblingNr

spaceafter

spacebefore

spacebetween

StartSpan

TableCount

Tables

8.2.2 TParagraph Methods

_FixAllCellWidths

_IsWidthTw

_SetMinMaxWidth

AddFlagAttr

ADelAllCharAttrDefinedIn

ADelColumn

ADeleteEqualSettings

AdjustTableRow

AGetBaseAndSpan

AGetFBBGColor

AGetInherited
AGetInheritedFromCell
AGetInheritedFromPar
ANeedUpdateProps
Append
AppendChar
AppendChild
AppendNewCell
AppendNewObject
AppendNewObjectPair
AppendNewPar
AppendNewRow
AppendNewTable
AppendTree
AsChildOfPreviousPar
ASetColumn
ASetNeutral
ASetRow
Assign
AUpdateProps
CalcBorderLeftRightIndent
CheckTable
ClearCharAttr
ClearCharAttributes
ClearProps
ClearText
Compare
ComparePar

CompareW
Contains
ContainsText
CopyChar
CPOfPos
Create
CreateCopy
CreateCopyList
CreateRow
CreateTable
CurrentSection
DeleteChar
DeleteChars
DeleteMarkedChar
DeleteParagraph
DeleteParagraphEnd
DeleteParagraphKeepChildren
DeleteWPTextObj
DelFlagAttr
Depth
Destroy
Duplicate
Empty
EndRow
Exchange
ExcludeProp
ExcludeProps
FirstLevelPar

FixAllCellWidths
FixAllRightCellWidths
FontSizeAtPosition
FreeObjectRefs
GetAllText
GetCell
GetCharAttr
GetCharAttrAt
GetCharAttrIndexAt
GetChildrenAsList
GetDebugTreeString
GetFirstCharAttr
GetLineText
GetNumberText
GetObjList
GetRowNext
GetRowPrior
GetRTLCursorPos
GetSubText
GetText
GetWPTextObj
globalnext
globalNextPar
HasAttr
HasObjects
HasParent
HasProp
HasText

HasTextW

Height

HeightTotal

InbetweenObjects

IncludeProp

IncludeProps

Insert

InsertEx

InsertNewObject

IsCharObject

IsEmpty

IsFirstPar

IsLastPar

IsLowercase

IsNonABC

IsNonSpace

IsNum

IsPunctuation

IsRTL

IsSpace

IsTable

IsUppercase

IsWordDelimiter

LastChar

LimitCPToLineLength

LimitLines

LineAllCount

LineEndOffset

LineHeight
LineIsLast
LineLength
LineOffset
LineOfPos
LinePageNr
LinePosFromX
LineStartNr
LineXFromPos
LoadFromFile
LoadFromStream
LoadFromString
LocateNextChar
LocatePrevChar
MakeLine
MergeCell
MoveChar
NeedAttr
NeedCharPos
nextcell
NextLine
Optimize
OptimizeObjectRefs
Overwrite
paprlsLeftPar
paprlsRightPar
ParagraphLines
ParagraphObjAdd

ParagraphObjDel
ParagraphObjFind
ParentCellFromTable
ParNr
Position
PositionOfObject
PosOfCP
prevcell
PrevLine
QuickFind
Reformat
Refresh
Replace
ReplaceW
RowAppend
SaveToStream
SetAllText
SetCapacity
SetCharAttr
SetChildrenFromList
SetRTFData
SetRTLCursorPos
SetStyle
SetText
SplitAt
SplitCell
SplitTable
StartNewSection

StartWith
 StartWithW
 SubPar
 SwapWithNextPar
 SwapWithPrevPar
 TableDepth
 TextAreaWidth
 TotalLength
 UnlinkParagraph
 UnlinkParagraphList
 UpdateMinMaxWidth
 ValidateTable

8.2.3 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;
    end;
  end;
```

```

    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).

8.3 TWPRTFDataCursor

The cursor class is responsible for cursor movement, text selection and the assignment of properties to the selected or current text. It is owned by the TWPRTFDataCollection. This means if different editor components share the same RTFDataCollection they also share the same cursor.

The class TWPRTFDataCursor 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. It also contains moving and selection methods. It offers access to the interfaces to change the attributes of the selected text or the current writing mode.

8.3.1 Drop-Markers

The 'Drop Markers' are special markers which can be placed in the text. They are not really inserted in the text (such as bookmarks). They are simple flags which can be used to quickly return to a certain position in the text. They are not saved with the text. Many of the input routines automatically move the drop marker, this makes them much more powerful than simply storing the text position (CPPosition).

```

function DropMarker: Integer;
function DropMarkerAt(par: TParagraph; PosInPar: Integer): Integer;

```

This functions drop a marker which can be collected later with CollectMarker. Text insertions and deletions - as long as no paragraphs are inserted or deleted - will will automatically modify the markers of the affected paragraphs to make sure their logical position is not changed. This makes markers much more powerful than cursor character offsets, such as 'CPPosition'.

We suggest to delete all markers with CollectAllMarker when they are not required anymore.

Note:

- a) that markers are not saved with the text.
- b) If the paragraph has been deleted which was used for a marker, the marker will be moved to the start of the following paragraph.

Drops a marker at the start/end of the selected text:

```

function DropMarkerAtSelStart: Integer;

```

function DropMarkerAtSelEnd: Integer;

Returns the text position of a certain marker. The result value will be -1 if the marker is not defined or if it is located in a RTFDataBlock which is currently not edited. (ActiveText) Please use GotoMarker to move the cursor to a certain position.

function DropMarkerPosition(DroppMarkerID: Integer): Integer;

Moves to a marker which was dropped by 'DropMarker'. If the parameter 'Collect' = TRUE this invalidates all markers which were dropped after the specified marker. You can use (true,-1) to collect the last marker or (true,X) to delete all markers including the marker X. (true,1) will collect all markers! If it is not possible to move to that marker the result value is FALSE.

**function GotoMarker(Collect: Boolean = TRUE;
DroppMarkerID: Integer = -1): Boolean;**

function GotoMarker(DroppMarkerID: Integer): Boolean;

Select the text between 2 markers.

function SelectMarker(FromMarker, ToMarker: Integer): Boolean;

Select text with a give length starting with a given marker.

function SelectMarkerStartLen(FromMarker, Length: Integer): Boolean;

Removes all markers from the text:

procedure CollectAllMarker;

Collects all markers including and after DroppMarkerID:

procedure CollectMarker(DroppMarkerID: Integer);

Example: This is the implementation of the ReplaceTokens procedure. It converts text which is wrapped by special characters such as "<<" and ">>" into mail merge fields, It uses the DropMarkers and the Finder. It also uses some 'Code' methods to create the objects which are used to mark mail merge fields.

```
function TWPCustomRtfEdit.ReplaceTokens(const opening, closing: string): Integer;
var
  s, r: string;
  StartID, SelStartID, SelEndID, rl: Integer;
  RestoreSel: Boolean;
  startf, endf: TWPTextObj;
begin
  StartID := 0;
  RestoreSel := FALSE;
  SelEndID := 0;
  SelStartID := 0;
  Result := 0;
  with TextCursor do
  try
    s := opening + '*' + closing;
    StartID := DropMarker;
    SelStartID := DropMarkerAtSelStart;
    SelEndID := DropMarkerAtSelEnd;
    RestoreSel := HideSelection;
    with Finder do
    begin
      ToStart;
      while Next(s) do
      begin
        r := FoundText;
        rl := FoundLength;
```

```

    CPosition := FoundPosition + r1 - Length(closing);
    DeleteChar(Length(closing));

    r := Copy(r, Length(opening) + 1,
        Length(r) - Length(opening) - Length(closing));

    endf := InputSingleCode(wpobjMergeField, r);
    endf.Mode := [wpobjUsedPairwise, wpobjIsClosing];
    // Move the cursor to the found position
    CPosition := FoundPosition;
    DeleteChar(Length(opening));

    startf := InputSingleCode(wpobjMergeField, r);
    startf.Mode := [wpobjUsedPairwise, wpobjIsOpening];
    endf.SetTag(startf.NewTag);

    MoveNext(Length(r));
    inc(Result);
end;
end;
finally
    HideSelection;
    if RestoreSel then
    begin
        TextCursor.SelectMarker(SelEndID, SelStartID);
    end;
    TextCursor.GotoMarker(StartID);
    Refresh;
end;
end;

```

8.3.2 Attribute Interfaces

The following references are accessible through the *TWPRTFDataCursor* class:

The interface to change the selected text. It is also available as *WPRichText.SelectedTextAttr*.
property SelectedTextAttr: TWPSelctedTextAttrInterface;

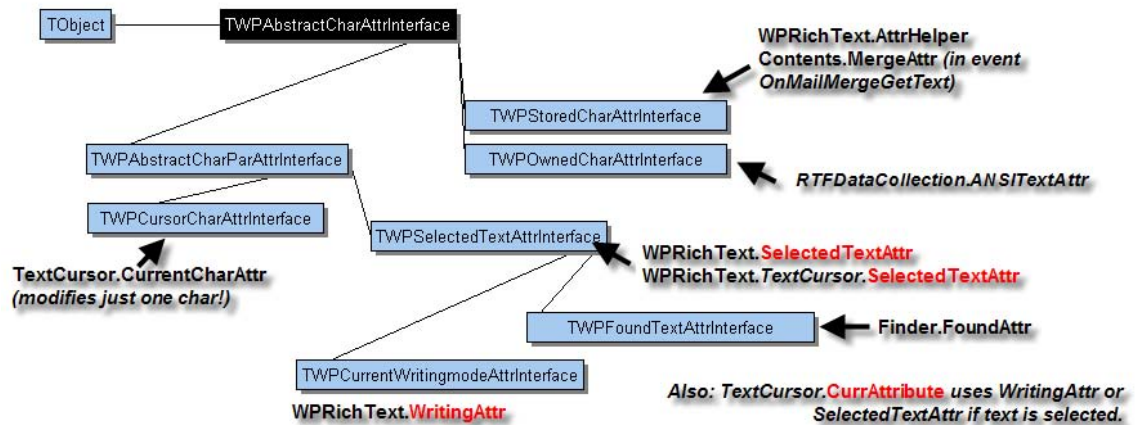
The interface to change the current writing mode. It is also available as *WPRichText.WritingAttr*.
property WritingTextAttr: TWPCurrentWritingmodeAttrInterface;

This interfaces changes the character atr the cursor position, similar to *WPRichText.CPAttr*.
property CurrentCharAttr: TWPCursorCharAttrInterface;

This function returns SelectedTextAttr if text is selected, otherwise WritingTextAttr:
function CurrAttribute: TWPSelctedTextAttrInterface;

Also available is an interface in the finder class to modify the found text.
property Finder.FoundAttr: TWPFfoundTextAttrInterface.

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 most important interface is **TWPAbstractCharParAttrInterface**. It defines the methods which are implemented by **TWPSavedTextAttrInterface** and **TWPCursorCharAttrInterface**. It also inherits of **TWPAbstractCharAttrInterface**.

Please note that the 'Get' functions return a boolean value. This value will be false if the property was not set. In this case the passed variable will not be modified.

If you work with colors you need to know that colors are saved as index values in the **TextColors** array. The methods with "Color" in their names automatically do the conversion.

In **TWPAbstractCharAttrInterface** the following methods are available to access the attributes of a character:

```

procedure Clear;
procedure Clear(const FontName: string;
    FontSize: Single; FontColor: TColor = clBlack); overload;
procedure BeginUpdate;
{ Saves the font name, color and background color and all other  
character attributes. Use AttrLoad to load.  
Note: The stored information survives a RTFProps.Clear! }
procedure AttrSave(var store: TWPAbstractCharAttrStore);
{ Loads the font name, color and background color and all other character  
attributes which were stored with AttrSave. If there  
are no stored attributes the result value is FALSE. }
function AttrLoad(var store: TWPAbstractCharAttrStore): Boolean;

{ This procedure assigns the character attributes stored in a certain text  
style or TParagraph to this attribute interface }
procedure Assign(SourceStyle: TWPTextStyle);
function EndUpdate: Boolean;
{ Applies style flags (such as WPSTY_BOLD..) to the character style. It is  
possible to switch a style explizietly off by setting the style bit  
in the "mask" and unsetting it in the "on" parameters. }
  
```

The following mask bits are available:

```

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
        xxx = 1024; // reserved bit
        WPSTY_DBLSTRIKEOUT = 2048; // Bit 12 strikeout - double solid line
        xxx = 4096; // reserved bit
        WPSTY_PROTECTED = 8192; // protected text

procedure SetCharStyles(WPSTY_mask, WPSTY_on: Integer);
// switches a style on and off
procedure ToggleCharstyle(WPSTY_code: Integer);

{ TOneWrtStyle is an enum which can be used to identify character styles.
  The following values are defined:
    afsBold, afsItalic, afsUnderline, afsStrikeOut,
    afsSuper, afsSub, afsHidden, afsUppercaseStyle,
    afsSmallCaps, afsLowercaseStyle, afsNoProof,
    afsDoubleStrikeOut, afsButton, afsProtected, afsUserdefine.
    Note: afsNoProof and afsButton are reserved for future versions.
  WrtStyle = set of TOneWrtStyle. }

function GetStyles(var styles: WrtStyle): Boolean;
procedure SetStyles(Value: WrtStyle);
procedure IncludeStyle(Element: TOneWrtStyle);
procedure ExcludeStyle(Element: TOneWrtStyle);
procedure UndefineStyle(Element: TOneWrtStyle);
procedure IncludeStyles(Elements: WrtStyle);
procedure ExcludeStyles(Elements: WrtStyle);
function HasStyle(Element: TOneWrtStyle; var Yes: Boolean): Boolean;
function HasAStyle(Element: TOneWrtStyle): Boolean;
procedure SetFont(const FontNr: Integer);
procedure SetFontName(const FontName: string);
{ reads the font name as index in the FontNames arra yof the TWPRTFdataProps }
function GetFont(var FontNr: Integer): Boolean;
function GetFontName(var FontName: TFontName): Boolean;
function GetFontCharset(var Charset: Integer): Boolean;
{ Assign the VCL style font flags. Better use IncludeStyle() and
  ExcludeStyle() }
procedure SetFontStyle(const FontStyle: TFontStyles);
{ Read the VCL style font flags. }
function GetFontStyle(var FontStyle: TFontStyles): Boolean;
procedure SetFontCharSet(const Charset: Integer);
function GetCharSpacing(var DistanceTW: Integer): Boolean;
procedure SetCharSpacing(const DistanceTW: 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 GetCharEffect(var Effect: Integer): Boolean;
procedure SetCharEffect(const Effect: Integer);
function GetCharStyleSheet(var StyleNr: Integer): Boolean;
{ WPTools can assign paragraph styles also to character attributes }
procedure SetCharStyleSheet(const StyleNr: Integer);
{ This function creates a CSS like style sheet using wptools special names.
  <br>If 'OnlyUsePTag' is set to true, only generic
  'P' tags are created: The created string can then only be used for text
  which uses the same RTFProps object in the same instance of the
  application. }
function AGetWPSS(OnlyUsePTag: Boolean = FALSE;
  Abbreviated: Boolean = FALSE): AnsiString;
{ 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.<br>
  If the optional parameter "Merge" is set to TRUE no properties in the
  style are initialized before the new properties are applied. }
procedure ASetWPSS(const WPCSSString: AnsiString;
  Merge: Boolean = false;
  Abbreviated: Boolean = FALSE);

{ This function returns true if the provided character attributes
  are contained completely in this character attributes.}
function Contains(var OtherCA: TWPCharAttr): Boolean;
function Contains(CharAttrInterface: TWPStoredCharAttrInterface): Boolean;
{ This function returns true if the provided character attributes
  contains all attributes which are defined here. }
function ContainedIn(var OtherCA: TWPCharAttr): Boolean;
function ContainedIn(CharAttrInterface: TWPStoredCharAttrInterface): Boolean;

```

TWPAbstractCharParAttrInterface defines this methods:

```

{ This procedure is used to set the paragraph style for all paragraphs }
function AGetBaseStyle(var StyleNr: Integer): Boolean;
{ This procedure is used to set the paragraph style for all paragraphs }
procedure ASetBaseStyle(StyleNr: Integer);
{ Increments a property (with offset>0) or decrements a property (with
  offset<0).
  In any case the value is checked for the minimum value }
procedure AInc(WPAT_Code: Byte; Offset: Integer; MinValue: Integer = 0);
{:: Assigns a color value - to reset to default use clNone }
procedure ASetColor(WPAT_Code: Byte; Value: TColor);
{ Executes a bitwise OR operation with the current value and the passed
  value and assign the result }
procedure ASetAdd(WPAT_Code: Byte; Value: Cardinal);
{ Executes a bitwise AND NOT operation with the current value and the passed
  value and assign the result }
procedure ASetDel(WPAT_Code: Byte; Value: Cardinal);
{ 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 }

```

```

procedure ASetNeutral(WPAT_Code: Byte; Value: Integer);
{ This procedure is used to reset a property to the default value }
procedure ADel(WPAT_Code: Byte);
{ This function returns the count of defined TabStops }
function TabstopCount: Integer;
{ This function returns all defined tabstops one by one }
procedure TabstopGet(nr: Integer; var Value: Integer;
    var Kind: TTabKind; var FillMode: TTabFill;
    var FillColor: Integer);
{ Adds a tabstop to the style. The value has to be specified in
  twips. If the function returns TRUE the tab was added, if
  it is false an existing was modified. }
function TabstopAdd(Value: Integer; Kind: TTabKind;
    Fill: TTabFill; ColorNr: Integer): Boolean;
{ Deletes the tabstop with the given value in twips + - the value RTFProps.
  TabPlusMinus) }
function TabstopDelete(Value: Integer): Boolean;
{ Moves a tabstop specified by position }
procedure TabstopMove(OldValue, NewValue: Integer);
{ Deletes all defined tabstops }
procedure TabstopClear;
{ Selects a base style for this paragraph. }
procedure SetStyle(ParStyleNr: Integer;
    ClearParStyles: Boolean = FALSE;
    ClearCharStyles: Boolean = FALSE);

{ Retrieves the number of the base style for this paragraph }
function GetStyle: Integer;
{ If this property is true all paragraph manipulations (ASet, AGet, ASetAdd,
  ASetDel, ASetColor, SetStyle, GetStyle but NOT the tabstop procedures)
  will be executed with the current cell instead of the current or selected
  paragraph. This makes it easier to change the color and border attributes in table
  cells.

The class TWPSSelectedTextAttrInterface does not use this property, only
TWPCursorCharAttrInterface. }
property ModifyCellsOnly: Boolean;

```

Examples:

a) Create formatted text under program control:

```

WPRichText1.AttrHelper.Clear;
WPRichText1.AttrHelper.SetFontName('Courier New');
WPRichText1.AttrHelper.SetColor(clGreen);
WPRichText1.ActiveParagraph.SetText(
    'Some green text',
    WPRichText1.AttrHelper.CharAttr);
WPRichText1.DelayedReformat;

```

b) Set the default writing font of a DBWPRichText. We use the OnClear event:

```

procedure TForm1.DBWPRichText1Clear(Sender: TObject);
begin
    DBWPRichText1.WritingAttr.Clear;
    DBWPRichText1.WritingAttr.SetFontName('Courier New');
    DBWPRichText1.WritingAttr.SetFontSize(18);
end;

```

c) Implement the hotkeys Ctrl+B, I and U to toggle the bold, italic and underline character style in

the current writing mode or, if text is selected, the selected text. We use the OnKeyPress event.

```

procedure TForm1.WPRichText1KeyPress(Sender: TObject; var Key: Char);
begin
  if Integer(Key) = Integer('B') - 64 then
    begin
      WPRichText1.TextCursor.CurrAttribute.ToggleCharstyle(WPSTY_BOLD);
      Key := #0;
    end
  else if Integer(Key) = Integer('I') - 64 then
    begin
      WPRichText1.TextCursor.CurrAttribute.ToggleCharstyle(WPSTY_ITALIC);
      Key := #0;
    end
  else if Integer(Key) = Integer('U') - 64 then
    begin
      WPRichText1.TextCursor.CurrAttribute.ToggleCharstyle(WPSTY_UNDERLINE);
      Key := #0;
    end;
end;

```

d) Change font size of one paragraph:

```

WPRichText1.SelectParagraph(); // You can pass a 'par' here!
if WPRichText1.SelectedTextAttr.GetFontSize(aSize) and (aSize < 11) then
  WPRichText1.SelectedTextAttr.SetFontSize(12)
else WPRichText1.SelectedTextAttr.SetFontSize(9);
WPRichText1.HideSelection;

```

e) Show all supported underline modes:

```

procedure TForm1.ShowPossibleULClick(Sender: TObject);
const
  cnames: array[1..18] of string =
  (
    'WPUND_Standard'
    , 'WPUND_Dotted'
    , 'WPUND_Dashed'
    , 'WPUND_Dashdotted'
    , 'WPUND_Dashdotdotted'
    , 'WPUND_Double'
    , 'WPUND_Heavywave'
    , 'WPUND_Longdashed'
    , 'WPUND_Thick'
    , 'WPUND_Thickdotted'
    , 'WPUND_Thickdashed'
    , 'WPUND_Thickdashdotted'
    , 'WPUND_Thickdashdotdotted'
    , 'WPUND_Thicklongdashed'
    , 'WPUND_Doublewave'
    , 'WPUND_WordUnderline'
    , 'WPUND_wave'
    , 'WPUND_curlyunderline');
var i: Integer;
begin
  WPRichText1.Clear;
  WPRichText1.CheckHasBody;

  WPRichText1.WritingAttr.Clear('Verdana', 10);

  WPRichText1.InputString('The modes are applied with' + #10 +
    'WPRichText1.WritingAttr.SetUnderlineMode(x)' + #13 + #13);

  WPRichText1.WritingAttr.ASet(WPAT_BorderFlags, WPBRD_DRAW_All4);

```

```

WPRichText1.WritingAttr.LockBorder;
WPRichText1.WritingAttr.ASet(WPAT_IndentLeft, 720);
WPRichText1.WritingAttr.ASet(WPAT_IndentFirst, 288);
WPRichText1.WritingAttr.ASet(WPAT_Indentrigh, 3600);
WPRichText1.InputString(#13);

WPRichText1.WritingAttr.SetColor(clGray);
WPRichText1.WritingAttr.SetUnderlineColor(clRed);

for i := 1 to 18 do
begin
  WPRichText1.WritingAttr.SetUnderlineMode(i);
  WPRichText1.InputString(cnames[i] + #13);
end;

WPRichText1.WritingAttr.SetUnderlineMode(-1);
WPRichText1.WritingAttr.SetUnderlineColor(clNone);

// After UnlockBorder the border will be closed with the next
// new paragraph
WPRichText1.WritingAttr.UnlockBorder;
WPRichText1.InputString(#13);

WPRichText1.WritingAttr.ADel(WPAT_IndentLeft);
WPRichText1.WritingAttr.ADel(WPAT_IndentFirst);
WPRichText1.WritingAttr.ADel(WPAT_Indentrigh);

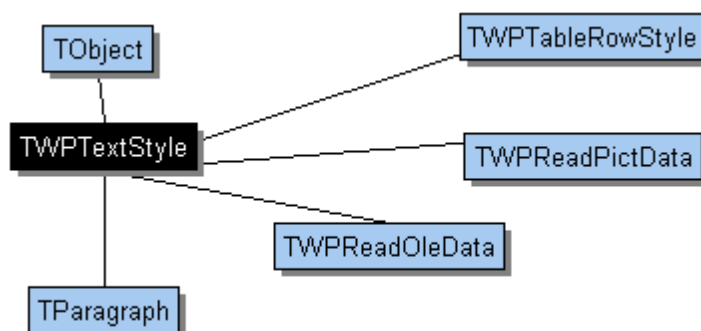
WPRichText1.WritingAttr.SetColor(clBlack);
WPRichText1.InputString(#13 + 'The colors have been changed with' + #10 +
  'WPRichText1.WritingAttr.SetColor(clGray);' + #10 +
  'WPRichText1.WritingAttr.SetUnderlineColor(clRed);' + #13 + #13);
end;

```

8.4 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](#).

The attributes of the `TWPTextStyle`/`TParagraph` objects are managed by the 'A' methods. Most important is the function `AGet` which reads a value.

This method `TWPTextStyle.AGet(WPAT_code : Integer; var Value : Integer)` is a function which returns a boolean value. The return code is false if the property with the ID `WPAT_code` was not

defined. In this case the variable "Value" will not be modified! Please make sure you initialized the variable Value!

8.4.1 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 inheritance 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 inheritance 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 properties 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.

8.4.2 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](#).

8.5 WPAT_codes

WPAT_codes are used to set and retrieve paragraph and style attributes.

Not all codes which are defined are already used in WPTools Version 7. The reserved codes are printed in gray color.

The method TWPTextStyle.**AGet**(WPAT_code : Integer; var Value : Integer) is mainly used to read a property. **Please make sure you initialized the variable "Value"!**

8.5.1 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
FLAG: WPEFF_CUSTOM1 = 1; // wpcustN - 63 Custom tags for whatever fixed styles you want to
develop
FLAG: WPEFF_CUSTOMMASK = 63; // The following are bits
FLAG: WPEFF_SHADOW = 64; // \shad
FLAG: WPEFF_INSET = 128; // \embo
FLAG: WPEFF_OUTSET = 256; // \impr
FLAG: WPEFF_OUTLINE = 512; // \outl
FLAG: WPEFF_FRAME = 1024; // \chbrdr - only default \brdrs\brdrw10
FLAG: WPEFF_ANIMbit1 = 2048; // \animtext1
FLAG: WPEFF_ANIMbit2 = 4096; // \animtext2
FLAG: WPEFF_ANIMbit3 = 8192; // \animtext4
FLAG: WPEFF_ANIMMask = 8192 + 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 ... )
FLAG: WPSTY_BOLD = 1; // Bit 1 bold
FLAG: WPSTY_ITALIC = 2; // Bit 2 italic
FLAG: WPSTY_UNDERLINE = 4; // Bit 3 underlined (solid)
FLAG: WPSTY_STRIKEOUT = 8; // Bit 4 strikeout
FLAG: WPSTY_SUPERSCRIPT = 16; // Bit 5 superscript
FLAG: WPSTY_SUBSCRIPT = 32; // Bit 6 subscript
FLAG: WPSTY_HIDDEN = 64; // Bit 7 hidden text
FLAG: WPSTY_UPPERCASE = 128; // Bit 8 all uppercase
FLAG: WPSTY_SMALLCAPS = 256; // Bit 9 all uppercase but non-captitals are 20% larger
FLAG: WPSTY_LOWERCASE = 512; // Bit 10 all lowercase
FLAG: WPSTY_NOPROOF = 1024; // Bit 11 \noproof - disable spellcheck for this
FLAG: WPSTY_DBLSTRIKEOUT = 2048; // Bit 12 strikeout - double solid line
FLAG: WPSTY_BUTTON = 4096; // button (use background color) - with frame
FLAG: WPSTY_PROTECTED = 8192; // protected text - can be optionally handled as shaded text
FLAG: 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

```

```

FLAG: cafsHyphen = $01000000; // assigned by reader or Ctrl+minus, break here

```

```

FLAG: cafsWasChecked = $02000000; // used by spellcheck routine - don't check again

```

```

FLAG: cafsMisSpelled = $04000000; // used by spellcheck routine - red line

```

```

FLAG: cafsMisSpelled2 = $08000000; // used by spellcheck routine - green line

```

```

FLAG: cafsInsertedText = $10000000; // WPTOOLS PREMIUM: Revision Marks

```

```

FLAG: cafsDeletedText = $20000000; // WPTOOLS PREMIUM: Revision Marks

```

```

FLAG: cafsWordHighlight = $40000000; // highlighted by Find Routine

```

```

FLAG: cafsDelete = $80000000; // Text is marked for deletion

```

```

FLAG: cafsALL = $FF000000;

```

```

FLAG: 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 ...

```

```

FLAG: WPUND_Standard = 1; // Underline Style 1

```

```

FLAG: WPUND_Dotted = 2;

```

```

FLAG: WPUND_Dashed = 3;

```

```

FLAG: WPUND_Dashdotted = 4;

```

```

FLAG: WPUND_Dashdotdotted = 5;

```

```

FLAG: WPUND_Double = 6;

```

```

FLAG: WPUND_Heavywave = 7;

```

```

FLAG: WPUND_Longdashed = 8;

```

```

FLAG: WPUND_Thick = 9;

```

```

FLAG: WPUND_Thickdotted = 10;

```

```

FLAG: WPUND_Thickdashed = 11;

```

```

FLAG: WPUND_Thickdashdotted = 12;

```

```

FLAG: WPUND_Thickdashdotdotted = 13;

```

```

FLAG: WPUND_Thicklongdashed = 14;

```

```

FLAG: WPUND_Doublewave = 15;

```

```

FLAG: WPUND_WordUnderline = 16;

```

```

FLAG: WPUND_wave = 17;

```

```

FLAG: WPUND_curlyunderline = 18; // only used for spellcheck

```

```

FLAG: WPUND_NoLine = 200; // Dont draw line !!!! When imported from RTF!
WPAT_UnderlineColor = 14; // Underlining color, 0=text color, otherwise colorindex +1
WPAT_TextLanguage = 15; // {reserved} Language of the text
WPAT_CharStyleSheet = 16; // CharacterStyle (index in ParStyles)

```

NOTE: **gray** symbols are reserved for future versions of WPTools!

8.5.2 Paragraph Attributes

```

// Margins
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

// padding, only in tables:
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
WPAT_WordSpacing = 28; // Value = 0 for default or % of EM (ignored for justified text)
WPAT_Alignment = 29; // paraLeft, ...
WPAT_VertAlignment = 30; // Cells: paraVertTop, paraVertCenter, paraVertBottom

// Colors and background
WPAT_BGColor = 50; // Background Color
WPAT_FGColor = 51; // Foreground Shading Color
WPAT_ShadingValue = 52; // Background Shading Percentage in %
WPAT_ShadingType = 53; // Background Shading Type
VALUE: WPSHAD_solidbg = 0; // Solid Background - use WPAT_BGColor and
WPAT_ShadingValue
VALUE: WPSHAD_solidfg = 1; // Solid Background - use WPAT_FGColor and
WPAT_ShadingValue
VALUE: WPSHAD_clear = 2; // Clear Background
VALUE: WPSHAD_bdiag = 3; // Backward diagonal pattern.
VALUE: WPSHAD_cross = 4; // Cross pattern.
VALUE: WPSHAD_dcross = 5; // Diagonal cross pattern.
VALUE: WPSHAD_dkbdiag = 6; // Dark backward diagonal pattern.
VALUE: WPSHAD_dkcross = 7; // Dark cross pattern.
VALUE: WPSHAD_dkdcross = 8; // Dark diagonal cross pattern.
VALUE: WPSHAD_dkfdiag = 9; // Dark forward diagonal pattern.
VALUE: WPSHAD_dkhor = 10; // Dark horizontal pattern.
VALUE: WPSHAD_dkvert = 11; // Dark vertical pattern.
VALUE: WPSHAD_fdiag = 12; // Forward diagonal pattern.
VALUE: WPSHAD_horiz = 13; // Horizontal pattern.
VALUE: WPSHAD_vert = 14; // Vertical pattern.
WPAT_BGBitMap = 54; // Background Image.
WPAT_BGBitMapMode = 55; // Bitfield for scroll, center, center, repeat

```

Special Flags:

```
WPAT_ParProtected = 92; // 1=protect, 0=not protected
WPAT_ParKeep = 93; // 1=no page break
WPAT_ParKeepN = 94; // 1=keep paragraphs together
WPAT_ParlsOutline = 160; // Is Outline (used by PDF Export and for TOC) - value = level!
```

NOTE: **gray** symbols are reserved for future versions of WPTools!

8.5.3 Numbering Attributes

```
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 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_WIDECCHAR = 15; // Double-byte character
{*}WPNUM_CHAR = 16; // Single-byte character
{*}WPNUM_CIRCLE = 19; // Circle numbering (*circenum)
{*}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
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
{*}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;

```

8.5.4 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) = BLeft
{*}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
```

8.5.5 Table Size and Position

A table usually uses the complete text area, this is the page width minus the left and right margins.

Property to set the width in twips: WPAT_BoxWidth

Property to set the width in percent * 100 of the page width
WPAT_BoxWidth_PC

Offset from the left margin. Can be negative to be to the left of the text area.
WPAT_BoxMarginLeft

Offset from the left margin. If negative the table is extended into the margin area

WPAT_BoxMarginRight

Horizontal Alignment of the table: WPAT_Box_Align:

possible values are WPBOXALIGN_RIGHT and WPBOXALIGN_HCENTERTEXT

The column width is defined by the properties **WPAT_ColWidth** and WPAT_ColWidth_PC. The first is the width in twips, the latter is the width in percent * 10 (Example: 1000 sets a width of 10%).

WPAT_ColWidth has priority over WPAT_ColWidth_PC. You will need to delete the WPAT_ColWidth flag if you need to set the width in percent.

```
Cell.ADel(WPAT_ColWidth)
Cell.ASet(WPAT_ColWidth_PC, 1000); // 10%
```

You can read the current width of a cell using the method cell.IsWidthTW. The text must be formatted. Once the user resizes columns in a table all widths which are defined by a percent value will be converted into exact WPAT_ColWidth values. You can force this conversion for the complete text using the function WPRichText.TableFixAllCellWidths. This method can also round the width values using a 'snap value'. The method TableAdjustCellWidth can be used to recalculate the width of all columns to keep them visible.

The property WPAT_BoxMinHeight can be used with table rows to set the minimum height of a table row. WPAT_BoxMaxHeight is used to set the maximum width. Both properties will be ignored in cells which are merged vertically. They can be used together.

You can use reference Cell.ParentRow to set or read a property in a row. The editor allows the modification of row heights if this mode has been enabled in the property EditOptions.

Example: Apply row height to all rows in current table

```
var par : TParagraph;

if WPRichText1.Table<>nil then
begin
  par := wp.Table.ChildPar;
  while par<>nil do
  begin
    par.ASet( WPAT_BoxMinHeight, twips_value );
    par.ASet( WPAT_BoxMaxHeight, twips_value );
    par := par.NextPar;
  end;
end;
```

Example: Apply row height to selected rows only

```
var par : TParagraph;

if WPRichText1.Table<>nil then
begin
  par := wp.Table.ChildPar;
  while par<>nil do
  begin
    if par.HasProp( paprCellIsSelected ) then
    begin
      par.ASet( WPAT_BoxMinHeight, twips_value );
      par.ASet( WPAT_BoxMaxHeight, twips_value );
    end;
    par := par.NextPar;
  end;
```

end;

9 C++Builder Notes

Installation

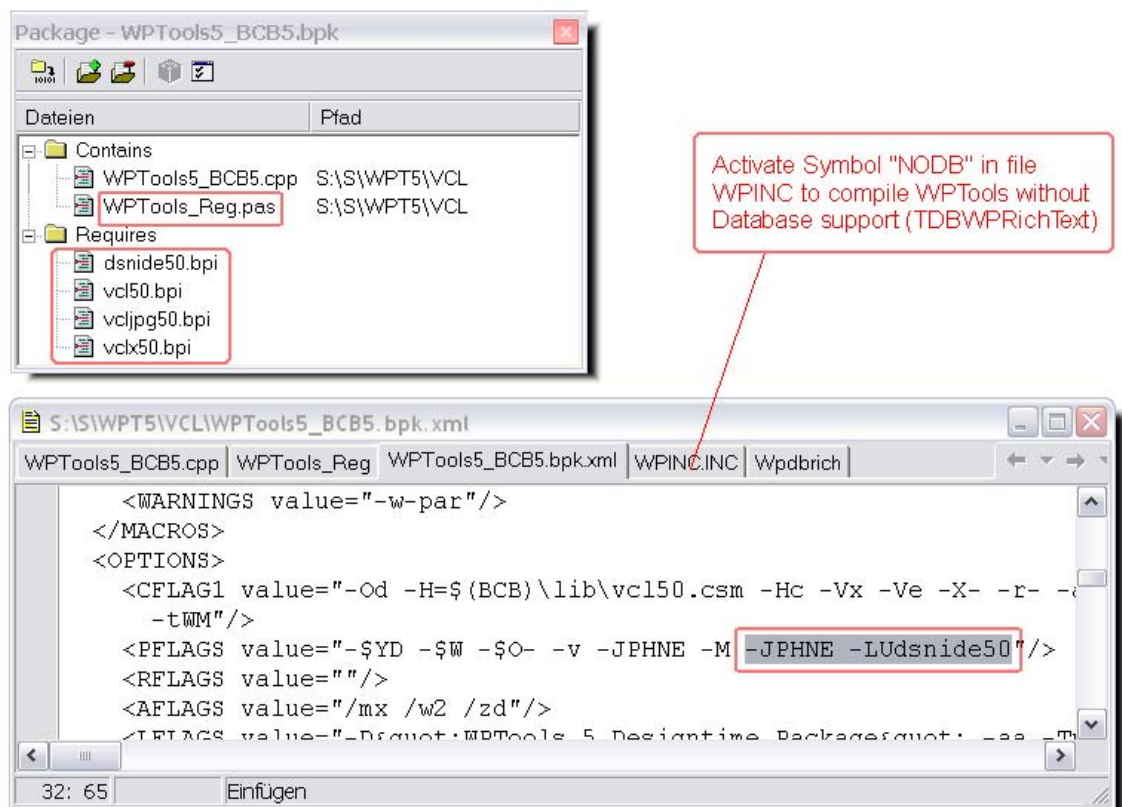
We added BPK files which can be compiled in C++Builder as packages. If there is a problem (Your setup is different than on the reference machine) you can install the file WPTools_Reg.PAS as a new component into a new package.

You will have to add the vcl and vclx package - otherwise the message "filename.obj was not found" will be displayed.

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.



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 ().

Speciality: Instead of TWPCustomRTFEdit.CreateDynamic use
edit = **new** TWPCustomRTFEdit(nil);
edit._MakeDynamic();

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.

9.1 Trouble Shooting/ C++Builder

To use the component with C++Builder 2009 or later please install the setup for the respective Delphi version.

In the package options please change the Linker option to create HPP files, LIBS, OBJs and HPP.

When creating a package with C++Builder 2007

Go to Project Options > Delphi Compiler > Other options

In the 'Additional options' box, enter
> -LUDesignIDE

Leave the 'Use these packages when compiling' box empty.

Compile.

This is described in the CB2007 help system. Search in the Index for "Delphi packages (C++)"

9.2 Example: Create dynamic TWPCustomRTFEdit

```
TWPCustomRtfEdit *DynRTFText;  
  
DynRTFText = new TWPCustomRtfEdit();  
DynRTFText->_MakeDynamic();  
  
DynRTFText->Clear();  
DynRTFText->Header->SetPageWH(WPCentimeterToTwips(21),
```

```

WPCentimeterToTwips(29.7),
WPCentimeterToTwips(2),    //left
Margin
WPCentimeterToTwips(2),    //right
Margin
WPCentimeterToTwips(2),    //top
Margin
WPCentimeterToTwips(2));    //
Bottom Margin

DynRTFText->CheckHasBody();
DynRTFText->InputString("Some Text\n");

```

9.3 Example: Create image object and insert

```

TWPOImage *image = new TWPOImage(0);
image->LoadFromFile( "logo.bmp" );

//image->PositionMode = wpotPage;

WPRichText1->TextObjects->Insert(image->CreateCopy(0), 0, 0,
"", "" );

image->LoadFromFile( "logo1.bmp" );
WPRichText1->TextObjects->Insert(image, 0, 0, "", "" );

```

10 Notes for Upgraders

10.1 ... when upgrading from WPTools 5 or 6

The RTF-Engine of WPTools 7 is based on WPTools 6 but has been reorganized.

There are the new units **WPRTEDefsConsts**, **WPRTEPlatform**, **WPRTEEdit** which usually have to be added manually to the project.

Further more it is necessary to add unit **WPRTEFormatA** and, if the AsWebpage view is used, also **WPRTEFormatB**.

Unless this units are added, the editor cannot display text. By default the unit **WPCtrMemo** links in both units - this can be switched off by disabling the **\$DEFINE WP_USEFORMATA** and / or **WP_USEFORMATB** in **WPINC.INC**.

Change of RTFEngine class architecture:

a) **TWPRTFDataCollectionBase** is now the basis **RTFDataCollection**.

b) The Type **TWPVirtPageRTFDataRef** was removed

This used to be a record. It has been replaced with **TWPVirtPageRTFDataFrame** class. It is now stored in a list and not an array. This makes it possible to access and store more data.

TWPVirtPageRect was removed

This is now a class TWPVirtPagePrintRect. The instances are stored in a TList.

Several TWPVirtPage "RTFData" functions have been modified to work with the new object class. function ElementPage has been removed.

d) DisabledDialogs

property DisabledDialogs: TWPCustomRtfEditDialogs was removed from the PreviewDialog and replaced by
property DisabledButtons : TWPPreviewButtons

e) The method RTFDataBlock.InsertPar was replaced by InsertParEx.

f) PastMode wpPasteSimlified was changed to wpPasteSimplified

10.2 ... when upgrading from WPTools 4

As emphasized before WPTools Version 7 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 Version 7 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 Version 7 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 is required to execute the procedure **ReformatAll;**

The same is required if you want to print or create a PDF file right after the use of LoadFromFile.

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!

Please note that the property **WorkOnText** used to select a completely different text (the header/footer text) in WPTools 4. In WPTools Version 7 it will simply move the cursor to that text. Especially the function MergeText normally works with the body text, the property WorkOnText will not make any difference. But the MergeText function to select the modification of all text parts.

WPTools Version 7 works with text attributes which can be undefined. So it is possible that the current font has no name. In this case the font used for the text will be defined by the paragraph style or the default attributes.

To set the default font for an editor use the OnClear event:

```
procedure TForm1.DBWPRichText1Clear(Sender: TObject);  
begin  
    DBWPRichText1.WritingAttr.Clear;  
    DBWPRichText1.WritingAttr.SetFontName('Arial');  
    DBWPRichText1.WritingAttr.SetFontSize(11);  
end;
```

Please note that the properties of the reader and writer (LoadOptions, StoreOptions) are now defined by [format strings](#). These **format strings** can be used with all IO methods. To retrieve the text as RTF string you can simply use `str := WPRichText.AsANSISString('RTF');`

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 Version 7 so please do not try to update the old logic to work with WPTools Version 7. 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 Version 7" 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](#)

10.2.1 Updated Rendering and Formatting

WPTools Version 7 formats as close to MS Word(tm) as possible.

So it ignores the paragraph spacing on top of pages. You can enable the old behavior in property FormatOptionsEx by adding flag `wpDontIgnoreSpacebeforeOnTopOfPage`.

Please also see the other options in FormatOptions and FormatOptionsEx.

10.2.2 Changed Unit Names

One of the first things you might realize when you try to compile a WPTools V4 project with WPTools Version 7, 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.

We recommend to delete all wp* units from the uses clause of each unit, add units WPRTEPaint and WPObj_Image and let Delphi add the other required units.

The following changes are required:

Change

WPDefs, WPObj, WPRtfTXT, WPRTFIO

all to **WPRTEDefs, WPRTEDefsConsts, WPRTEPlatform, WPRTEEdit**

Change

WPRtfIO, WPRtfInp

to **WPRTEPaint**

Change

WPEmOBJ

to **WPObj_Image**

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 **WPCTRMemo**

The unit WPCTRMemo contains the editor control and the preview, making unit WPPreVw obsolete.

Change

WPRich

to **WPCTRRich**

The unit WPCTRich 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 unit WPStat2 is not included.

Please use a standard status bar (Win32 tab). The old WPTools 1 status bar has been discontinued.

The HTML support is included in unit **WPIOHTML**, the units WPreadHT, WPWrtHT and WPXMLFile are obsolete. WPTools Version 7 will always use CSS in the created HTML code.

The RTF reader is implemented in unit WPIOReadRTF, the RTF writer in WPIOWriteRTF. (WPreader and WPWrtRTF is obsolete)

The ANSI reader and writer are implemented in WPIOANSI, not WPWrite2.

Important: You need to use "[FormatStrings](#)" instead of the properties LoadOptions and StoreOptions!

unit WPPapBin always was nothing more than an example unit. The old version will still work. Also see demo: "PrinterSetup"

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

The component WPFileer is not supported anymore.

Quickreport support has been discontinued.

10.2.3 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 Version 7 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-Engine 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.

10.2.4 Changed Pointers

To make compilation with Delphi8 for .NET possible, WPTools Version 7 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 7 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 Version 7 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.

10.2.5 Changed GUI elements

TWPToolCtrl

In WPTools 4 the control TWPToolCtrl 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 \(Toolbar2000 Extension\)](#)

TWPEdit

The component TWPEdit (the single line RTF object) is not available anymore.

You can use a standard TWPRichText and set certain properties to replace it.

```
object WPRichText1: TWPRichText
  LayoutMode = wplayNormal
  ScrollBars = ssNone
  EditOptions = [wpActivateUndo, wpActivateUndoHotkey, wpNoVertScrolling]
  XOffset = 15
  Width = 300
  Height = 22
end
```

Now add an event handler for OnKeyDown to suppress the Return key:

```
procedure TForm1.WPRichText1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key = VK_RETURN then key := 0;
end;
```

10.2.6 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.

10.2.7 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.

Please read [here](#) about how to use the paragraph styles in a native way.

10.2.8 BackgroundImage

WPTools Version 7 does not have the property "background image" anymore.

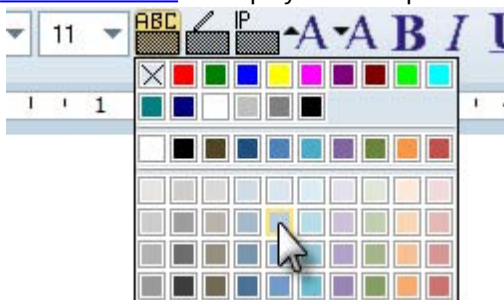
Instead you can draw the image using the water mark event - see Demo '[WaterM2](#)'. You can create a different tiled background on each page and it is also possible to show a form in the background. WPTools Version 7 takes care about the necessary buffering so no flickering will be visible.

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

11 Release Notes - WPTools 7

18.5.2013 - WPTools 7.01

- + ConvertTablesToText, convert all tables to text.
- the new GDI+ based image drawing code was not supported by printers
- fix minor drawing problem in Toolbar configuration
- + new component [TWPCreatePDFDlg](#) (can be used with wPDF V3 or later)
- + [Toolbar buttons](#) to display color drop down: (property WPToolbar.StandardColorDropdowns=false)



- + Style and Paragraph Border dialog now used the new color selector

13.5.2013 - WPTools 7.00

- + Using the global variable [WPDRAWRECTWITHBITMAP_BITMAP](#) the shading can be customized.
- * Alnc now uses default font size
- replaced OnTextNotFound with OnSearchReplaceMessage which not only by the find method but also bei the ReplaceMethod.
- * TWPRTFEnginePaint -> TWPRTFEngineEdit, requires unit WPRTEEdit

- * wpDefActions ---> wpDefActions7
- + CopyToClipboardAttr
- + BrushStart
- + CPMoveToItem
- + TWPParPopupObj
- + visual Style Picker component (tiles with style name and preview)
- + TRTFProps now component. Can be assigned to WPRTFPropsComponent
- + PageMinCount
- + PageMaxCount
- + updated WPGutter to display style used by text
- * InsertGraphic with parameter AsLink=true will trigger the event OnRequestHTTPImage with Reader=nil
- + Paragraph.LoadFromString has new options
- + WPRichText1.FindTextBoxPar can create sub paragraphs
- + event: RTFData.AfterChangeTextBoxPar
- + Labelprinting: Caption and frame is not painted for empty labels. wpfLabelAllowSoftPagebreak
- + images can be embedded into HTML
- + ParStyles.SetCSS(...)
- + Themed Gutter - use clNone as background color
- + CombineCellsVertically improved
- + Added API calls to TParagraph: ASetRecursive
- + save "variable" in WPIOWPtools
- + improved theme support (ruler, editor with scrollbars, toolbar, Gutter)
- + new Ruler Design and Symbols (Shaded)
- + internal FIND Dialog
- + internal Replace Dialog -- for better theming
- + WPLoremIpsum
- + ReplaceTextMethod
- + CopyAttr
- + PasteAttr

formatting now done in

WPRTEReformatA - WTools 6 formatting (old logic but adapted to use new datastructures)

WPRTEReformatB - WTools 6 HTML formatting (old logic but adapted to use new datastructures)

- RTFDataCollection does not implement TextObjects anymore. It is implemented in WPRTEEditor

- TWPRTFDataCollectionBase is now the basis RTFDataCollection.
We moved edit parts to WPRTEEdit. There is also the merge code.
TWPRTFDataBlocked

- new unit: WPRTEDefsConsts

- new unit: WPRTEPlatform

- new unit: WPRTEEdit

*** Changes to the Engine types and logic ***

a)

type TWPVirtPageRTFDataRef was removed

This used to be a record. It has been replaced with TWPVirtPageRTFDataFrame class.

It is now stored in a list and not an array. This makes it possible to access and store more data.

TWPVirtPageRect was removed

This is now a class TWPVirtPagePrintRect. The instances are stored in a TList.

Several TWPVirtPage "RTFData" functions have been modified to work with the new object class.
function ElementPage has been removed.

b)

property DisabledDialogs: TWPCustomRtfEditDialogs was removed from the PreviewDialog and replaced by
property DisabledButtons : TWPPreviewButtons

ButtonHeight in Toolbar must be set. Default 22.

RTFDataBlock.InsertPar was replaced by InsertParEx.

12 Release Notes - WPTools 6

We include the WPTools 6 release notes here to provide You with hints to interesting changes and improvements.

Release Notes:

19.2.2013 - WPTools 6.29.1

- fix in rtf writing code to solve problem with merged cells
 - fix possible rangecheck error
 - fix problem with TextObject.LoadFromFile and Delphi XE3
 - * RTF reader now handles UNC file links which use "\\\" in the path
 - * the cursor was not painted if DoubleBuffered was set to true for the parent of the editor
 - + WPTools Premium: Saves and loads \column
 - * improved theming of TWPToolbar and TWPToolPanel
 - + new event: OnPaintDesktopBackground. It can be used to draw the parent of the editor, for example if it is a TMS panel or pager control.
- Example:

```
procedure TForm1.WPRichText1PaintDesktopBackground(Sender: TObject; Canvas: TCanvas; R:
TRect);
begin
    // This would paint the TWPRichText, too - but TWPRichText is locked for repaint during this event
    AdvOfficePager1.PaintTo(Canvas, -WPRichText1.Left, -WPRichText1.Top);
end;
- HighlightTextColor can now also be used if 2Pass Painting is used
```

21.12.2012 - WPTools 6.29

- images in RTF label were not painted when label was moved
- + added support for XE3 to WPTools STD edition
- * stream RTFvariables were not loaded from WPT format. They are loaded now.

9.11.2012 - WPTools 6.28

- Update to RTF reader to load landscape flag for sections better
- when page mirror was used, after a page break the text indentation was sometimes wrong
- hyphenation code was broken

- workaround for word files which have space characters in table definitions

16.10.2012 - WPTools 6.27''

- * some additions to the PRO edition for XE3

26.9.2012 -WPTools 6.27'

The PRO Version now supports Delphi XE3

3.8.2012 -WPTools 6.27

- fix for wrong display of tables with header and footer rows. Sometimes both wer painted without any data.
- + to load old Hiedit templates as RTF code use the formatstring -HiEditFields. This will create merge fields for ALL fields.
- NL sign was not shown right after CTRL+ENTER was pressed (requires ShowNL)
- fix for rangecheck exception with paintpages array
- fix for footer and page mirror
- doubleclick word selection now stops at NL
- Workaround for Windows Spooler problems - some images would get lost
- sections use footer and header of previous section, not general
- ASetBorder did change all border types

12.3.2012 -WPTools 6.25.4

- * allows changing of column width in redonly editors. Can be switchoed off in EditOptions or set compiler define TOTALREADONLY
- + wpDisableSelectAll in EditOptionsEx2
- * changed reformat/repaint after Undo/Redo
- pro and premium: Due to a problem with precompiler cursoir selection did not work correctly

8.3.2012 -WPTools 6.25.3'

- borders for paragraphs with multiple lines were not drawn correctly
- change in DBWPRich.pas to use LoadFromString instead of Text
- fix possible range check error
- change in WPTbar.pas to use different default for BevelOuter
- change in WPIOHTML to use default charattr of paragraph is a paragraph is empty

9.2.2012 -WPTools 6.25.2

- * new 2-pass painting triggers CustomPaint event only on second loop (when the text is painted)
- * changed protection of empty paragraph to WPTools 5 way
- + inside of the OnPrepareImageForSaving event it is now possible to set Writer.CurrentImageName to the name of the file which should be saved. This is only useful if ObjRef = nil and so no ObjRef. Filename can be set.
- + TParagraph.GetSubText now has optional parameter to disable the object reference char codes #1, #2 and #3
- * HTML writer ignores #13 codes when writing the text.
- * additional savety code in HTML writer
- * HTML reader sets image width and height to contents, (if possible)
- * HTML reader: changes UTF8 handling for UTF8Sequence=1
- * HTML reader does not stop on \0 anymore
- * HTML writer writes img tag also for empty images IF a name is provided in event PrepareImageforSaving
- * HTML reader does not add space at end anymore
- the UNICODE reader uses attribute of current paragraph. This is important for consistent behaviour between ANSI and UNICODE version
- when writing HTML background color is not set to white when shading is 0. 0 is treated as default value.
- image align center and right now works in HTML
- fix an endelss loop when image was too large
- improvement of table border drawing
- improvement for right align in table cells
- + numbering will be used when wpFormatAsWebpage was set in AsWebpage

10.1.2012 -WPTools 6.25

- + HTML reader reads cell heights
- RTF writer writes background color easier to understand by Word
- * improved XML reader/writer (unit WPIOXml1)
- * improved word wise cursor movement when fields are used
- + new "paint attributes" mode.
Use WPRichText1.BrushStart to select this mode.
- dashes were not painted using the current font color
- some stability improvements

7.11.2011 -WPTools 6.22

- + procedure TParagraph.CellSelectionAdd;
- + procedure CellSelectionRemove;
- + EditOptionsEx2: wpCellMultiSelect - allows multiselection in tables when CTRL is pressed
- + improved XML import/export (unit WPIOXML1.PAS)
- some smaller bugs fixed

3.11.2011 -WPTools 6.21.2

- fix problem with TWPToolButton
- improved HTML writer to write parameters in ""
- improved display of arabic text

24.10.2011 -WPTools 6.21.1

- fix problem when painting insertpoints after tab stops. They were painted two times.
- fix in XML and HTML writer. Close <div> tags when extracting text from fields

19.10.2011 -WPTools 6.21

- + CodeLocate can now also accessed in non-visual TWPRTFDataCollection and not just TWPCustomRtfEdit
- + CodeSelect can now also accessed in non-visual TWPRTFDataCollection and not just TWPCustomRtfEdit
- + TWPRTFDataCollection.CodeLocatePair(FormatName : String; var spar, epar : TParagraph; var spos, epos, : Integer) :Integer;
- solves problem with integrated Bin64 decoder
- + DeleteField now has optional "Contents" parameter to delete a certain field with contents
- + the text writer now understand the option -softlinebreaks to create a \n at the end of every line. In fact all soft line breaks will be handled like the #10 code.
- + CodeLoadSaveEmbeddedText - load or save text in fields, bookmarks, hyperlinks
- + the regular save and load methods (LoadFromFile, SaveToFile) can now access text wrapped by paired objects.
specify the fieldname in the format string, i.e. "f:name=RTF" to save or load the contents of the field "name".
- + There is an overloaded LoadFromString which expects a WideString as parameter (Delphi 6+)
- The Setup named the Delphi 2009 files "Delphi 2005" due to a typo. (Delphi 2005 units are not included anymore)
- fix probable range check error in WPRTDEFS

13.10.2011 -WPTools 6.20

- + completely new setup procedure. The PRO and Premium releases don't include object files which makes them much smaller.
- * compiled newWPTools 6 Reference (CHM file)
- * Delphi XE2: several small changes to improve theming support
- * Delphi XE2: several small changes to provide compatibility to 64bit compiler (requires WPTools PRO)
- + new demo developed with Delphi XE2, showcases actions, splitscreen, simulated MDI and property dialogs (Demos\XE2\WPWord)
- + wpDeleteAtEOFRemovesSpaceAfter in EditOptionsEx2
- TextObjectsGetList did not work
- * WPRuler uses Delphi XE2 VCL Theme plus
- * added defaults to properties TWPRichText

- change in RTF reader to let section inherit the default layout, not the current page layout
- fix of problem with table borders when also PageMirror was used.
- * change to DeleteMarkedChar. It now has additional parameter DeleteEmptyParAndTables : Boolean
- * change in unit WPWordConv to handle RTF as DOC files if they do not start with "{\rtf"
- * updated border dialog TWPParagraphBorderDlgEx
- * updated border drawing code - now supports dotted lines with wider lines.
- * modified method DeleteColumn
- * modified WPT reading code to repair table width which were negative
- + improved image rendering code for transparent (PNG) images. They will be drawn transparently also when scaled and also in high resolution rendering mode.
- + new code to draw dotted lines which also supports wider lines
- + new function WPPReplaceTokens (unit WPUtils.PAS). It is a ReplaceTokens function to be used on a TWPRTFDataCollection, not TWPRichText
- WPPremium - fix problem when there were too columns
- * MergeText now restores before Merge Cursor position and selection (except for cell selection)
- * resizing a table column does not move the cursor to the nearby cell anymore
- * different frame line when resizing columns and rows
- + InsertColumn now also works if wpAllowSplitOfCombinedCellsOnly was used in EditOptionsEx
- + new event OnPaintTextFrameRect let you paint background and borders for text frames, i.e. the text body or, with WPTools Premium, custom frames.
- + WPPREMIUM: In OnMeasureTextPage it is possible to set columns for certain pages.
Using PageInfo.colflags=1 it is possible to activate a line between the columns
It is also possible to add custom frames using PageInfo.rtfpage.AddFrameRect.
- + new ViewOptionEx: wpHideParBordersBeforeAndAfterPageBreaks
- + improved paint routine now avoids clipping of characters which were overlapping their bounding box, such as italic letters or "f".
The improvement is especially visible when selecting text or using character background colors
- + WPPREMIUM: it is now possible to print a line between columns using wpShowColumnCenterLine in ViewOptionsEx
- + With WPTools Premium it is now possible to print a line between certain columns -
use par.ASet(WPAT_COLFLAGS, WPCOLUM_LINE);
- + paragraph styles can now contain border definition for paragraphs
- + TWPTTextObjList now has a IndexOfSource function and a Source[] string array to access the objects
- * revised code to draw double borders - always draws two lines on screen even when zoomed
- * improved saving of numbering attributes with styles
- * style dialog can now apply number level even if style does not have numbering yet.
- * revised wpNoEditOutsideTable - was not checked for keyboard input
- * fix problem with - - - - at end of line
- fix problem with spell-as-you go after hyperlinks
- fix problem with page numbers in sections when tables were spanning pages
- fix problem with right aligned negative numbers in merge fields
- * automatic text attribute was not inherited to tables inserted in fields
- * images with mode "under text " can now be also clicked at positions where there is no text.
- WPLngCtr now defines DONT_OVERRIDE_COM, that fixes the IDE problem with DevExpress Toolbar controls

18.7.2011 -WPTools 6.16.2

- * ObjectMode wpobjPositionAtCenter can be used to align character based images to the center line. (Other RTF reader will not understand this feature)
- * some changes to prepare 64bit compatibility (requires WPTools 6 PRO)

7.7.2011 -WPTools 6.16.1

- * change to avoid flickering when doing auto scroll
- * message strings are now loaded from resource strings
- modified DBCS support for RTF reader
- * outerborder action now works for single cells and paragraphs

15.6.2011 -WPTools 6.16

- + using WPRichText1.Caret.Blink:=true it is possible to activate the blinking caret (0.5sec interval).
- * updated actions to apply inner and outer borders to selected cells
- update to ExSymbolDialog, Tab and Table dialog
- improved WPTOOLS reader to read and apply outline properties to current outline setting (unless

wpLoadDoesNotOverride is used)

- * tripple click in margin selects paragraph
- + double click in margin selects current cell
- + tripple click in margin selects current row
- change for PaintEngine and TWPRichTextLabel to not shrink tables which are small enough to fit the page

This solves a problem with dissapearing text in WPRichTextLabel

- fix selection problem when several images were linked to same paragraph
- when moving images the Z order will not be reset
- HTML Writer: A style with name "DIV" will be added to the style sheet to save the default font
- HTML Writer: BaseFont tag will now be written with font size (requires -writebasefont option)
- improved display of character background color for fields and other special code

8.5.2011 -WPTools 6.15

- * updated border painting
- * updated Inner/Outer Border Action
- * new object sizing routine lets the user make the size larger than the page
- update in WPTools reader to overwrite outline styles when loading tzhe same group

9.3.2011 -WPTools 6.14.6

- * change in format routine to fix problem when a nested table cell caused a page break.

14.12.2010 -WPTools 6.14

- * fix for SetAsString code (Unicode Delphi)
- * several fixes and updates in editor
- * WPTools Premium: \$define DONT_AUTOENTER_TEXTBOXES in WPINC.INC to switch of the behavior, that when editing a textbox the user can click on any other and edit that.

22.9.2010 -WPTools 6.13.3

- * WPCtrMemo.PAS now defines TEXT_IS_UNICODE fro Delphi 2009 and later. Now the property Text and SelText reads and writes unicode strigs
- * change in RTF reader to read ANSI characters in the range 128..255
- * Tables and rows can now be hidden. (TParagraph.Hidden)
- * The Lines property now supports unicode strings (Delphi 2009 and later)
- + HTML reader and writer now use the entitly ­ as soft hyphen

27.8.2010 -WPTools 6.13.2a

- + new ViewOptionEx wpUnderlineWebLinks. If active links like <http://www.wptools.de> will be drawn using the attributes for hyperlinks
- The HyperlinkCursor will be selected and the hyperlink event will be triggered
- * other fixes in RTF engine
- Memo._OverrideCharset was not set to -1

27.7.2010 -WPTools 6.13.1

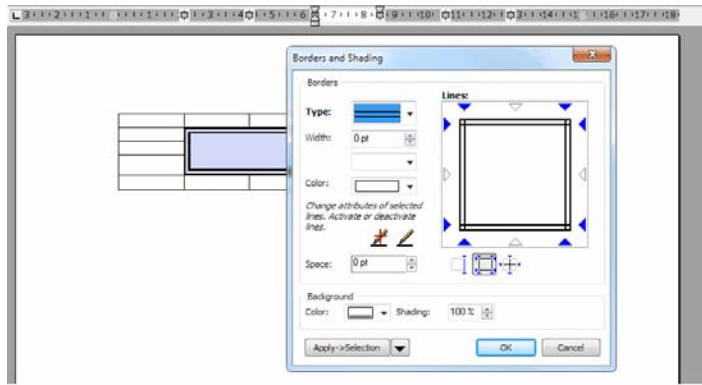
- + WPRichText1.Memo.ColorGridLines can be used to change the color of the grid lines (ViewOptions)

23.7.2010 -WPTools 6.13

- * several improvements of editor
- * improvement to RTF writer to when writing table cells
- * improved right aligned text
- * fixed problem with line heights of lines which are empty except for new line character

17.6.2010 -WPTools 6.12.1

- + all new, powerful yet intuitive Border dialog.



The new extended border dialog added to WPTools 6.12

The border dialog can modify a range of selected cells, columns, rows, tables and also only modify certain properties while leaving the others unchanged.

You need to activate the compiler symbol NEWBORDER to use it by default

```
+ new method: procedure SetBorderProperties(Inner, Outer: TWPTTextStyle;
    ApplyMode : TWPParagraphBordApply;
    Mode : TWPUUpdateBorderProperties =
    [wpSetBorderFlags, wpSetBorderType, wpSetBorderWidth, wpSetBorderColor, wpSetParColor,
wpSetParShading]);
```

This method is mainly used by the new border dialog.

+ new flags in ViewOptionsEx:

wpShowCurrentCellAsSelected, // Displays current cell to be selected. Disables current selection

wpShowCurrentRowAsSelected, // Displays current table row to be selected. Disables current selection

wpShowCurrentTableAsSelected // Displays current table to be selected. Disables current selection

+ new property YOffsetNormal to defined an upper border for normal and wordwrap view.

* feature Header.MarginMirror changed to work like MS Word

new flag: wpMarginMirrorBookPrint in FormatOptionsEx2 to enable the previous logic

* workaround for MouseWheel UP beeing triggered too often (10 mms check)

+ ClipboardOption wpcoDontPasteHTML to disable HTML pasting completely (avoid problems with firefox)

+ it is now possible to load base64 embedded JPEGs from HTML

* it is now possible to change width of tables which exceed right margin

- fix bug in HTML writer for lists in table cells

- fix in RTF writer to write character colors also for text which is using a character style

- fix: numbering was not always updated

- fix: better use fonts in certain RTF files written by MS Word

* the new border dialog now reads the current border attributes from table cells, tables or selections

- Premium: fix problem when loading columns which started on first line

- fix in wpfUseKerning mode (FormatOptionsEx2) - it did not work as expected with some texts

(We recommend to use wpfUseKerning - it produces better print quality on screen)

* Pasting of HTML now works better

6.5.2010 -WPTools 6.11.2

* improvement to border rendering

* improvement to XML unit WPIOXML1 (Premium)

5.5.2010 -WPTools 6.11.1

+ ConvertTableToText now supports option to also handle soft line breaks

- fix problems with underlines at end of line

- fix problem when loading hyperlinks in RTF

- fix problem when saving attributes to XML (WPTools Premium)

- fix problem with text rendering

- fix problem with tables which have header row and pages with different header margin

19.4.2010 -WPTools 6.11

+ EditOptionEx: wpRowMultiSelect

+ new event: OnInternPaintPar

+ new event: RTFDataCollection AfterApplyUndoObject

14.4.2010 -WPTools 6.10.6

- * fields are now passed as unicode strings to PDF exporter
- * Delphi 2010/2009 import has been improved to load unicode values which are stored in fields.

5.4.2010 -WPTools 6.10.5

- + flag: wpDontExtendSelectionToRightMargin. Do not extend selection to end of line
- + wpInvertActiveRow in ViewOptionsEx
- some fixes for Delphi 2009 and Delphi 2010
- fix for section support
- WPTools premium: Fix for images in text boxes
- workaround to load RTF which use emfblib for pngblib

28.2.2010 -WPTools 6.10

- improve word left/right movement to skip hidden text
- improve http load of images
- improve support for numbering
- improve saving of character style attributes

11.2.2010 -WPTools 6.09.1

- + LoadFromString now has a "WithClear" parameter
- fix in RTF reader to better load files which do not define codepage
- fix in paint routine to solve a rare lockup
- fix in format routine to improve section support

6.2.2010 -WPTools 6.09.1

- fix of problem in save routine when footnotes were used (WPTools premium)
- fix in HTML writer
- Image options now have a Rotation property which allows 90, 180 and 270 setting.
- HTML reader and writer now support different colors for left, right, top, bottom lines

1.2.2010 -WPTools 6.09

- HTML writer will write 8 hard spaces for TAB stops at the start of a paragraph
- HTML writer will write page information only if -PageInfo was used in format string
- fix problem with left aligned text and image wrap around (wrong alignment)
- fix problem with sometimes duplicated images in PDF export
- fix problem with black rectangle in first line under Windows 7, 64 bit
- Improvement to RTF reader to ignore section properties which are not followed by \sect

14.12.2009 -WPTools 6.08

- graphics are resized to fill text area
- fixed problems in numbering
- fixed problem with one word paragraphs in justified paragraphs
- other improvements in editor

26.10.2009 -WPTools 6.07

- * improved layout of most important dialogs
- * improved extended insert symbol dialog
- fix in RTF reader to load sections and header+footer written by Word 2003
- don't add unwanted cell padding when loading table cells
- fix in WPTools reader to read custom number styles
- if paragraph styles use number styles the indent defined in the style has priority over numberstyle
- LabelDef now also works for one row and one column
- better handling of mousewheel event
- fix for tabs in tables
- + GIF animation (requires GifImage) library (not threaded)
to use it You need to set ViewOption wpUseOwnDoubleBuffer and call the method RefreshAniImages using a timer object.
- fix problem when sections were used with LabelDef.Active = true
- * change in HTML writer to close tags before paragraph end

4.10.2009 -WPTools 6.06

- fix problem with Delphi 2010 support (language control)
- fix problem with PDF export to reduce PDF size

- improve support for IME
- * improve AsWebPage format mode. Now WordWrap property is supported.
- fix searching text upwards with "Whole Word" selected

3.8.2009 -WPTools 6.05.9

- + added Delphi 2010 Support

3.8.2009 -WPTools 6.05.8

- * when using "Delete All" in the tabstop dialog, all tabs will be cleared
added to manual: Tabstop Category
- fixed problem when deleting text in a paragraph. The alignment was cleared unexpectedly.
- fix problem with installer, WPMangeHeaderFooter.DFM was not included
- fix for IPara in mail merge field objects
- improved handling of hover effect for hyperlinks
- improved text rendering for wPDF output (CID Mode)
- add correct WPManHeadFoot.dfm

23.7.2009 -WPTools 6.05.7

- + WRITE_PROP_WPTOOLSOBJ \$define in WPIOWriteRTF. Avoid problems when saving RTF and opting in Word
In case of special objects, such as SPAN codes, *\wpfldinst is beeing written what is ignored by Word
- + Dialog HeaderFooter can optionally create and manage header&footer for the current section
- + new KeepN Handling. This is by default activated in FormatOptionsEx2
- + new wpfHideParagraphWithHiddenText in FormatOptionsEx2.
Now paragraphs will be hidden if empty or only contain hidden text.
- + new format option -zap1 will remove the every first byte to convert a two byte stream into singly byte
-zap2 will remove every second byte. Usie this option when loading data from unicode data sets
- bugfix for table loading in RTF

15.7.2009 -WPTools 6.05

- + Scroll with middle mouse button

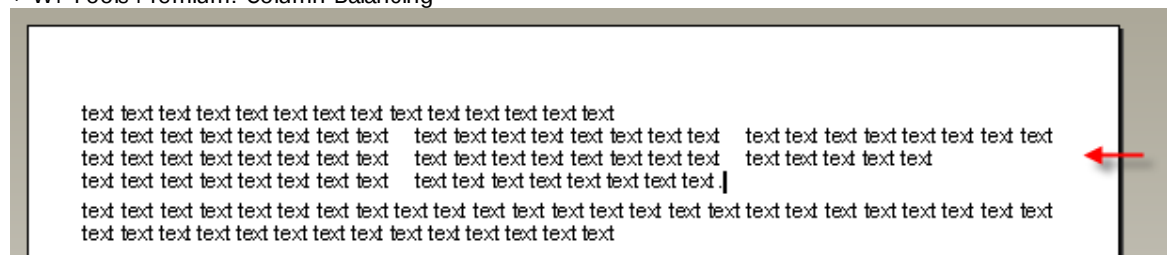


(when middle mouse button is pressed)

- + new XML editing mode (see [XML editor mode](#))
- + TParagraph.Trim method to remove white spaces at start and end
- + Vertical Scrolling by pressing the middle mouse button now works.
- + improved auto thumbnail mode
- * enhancement to HTML reader / writer to handle embedded SPAN objects
- + new method: ApplySPANStyles(and_remove : Boolean=false; ignore_charattr : Boolean = false);
can be used to apply SPAN styles to the text which it embeds
- + The function InputSpanObjects(Attributes : TWPAbstractCharAttrInterface) : TWPTTextObj;
can be used to wrap the selected text into SPAN objects
- + method LoadCSSheet can be used to load paragraph styles in CSS format from a string. There is also SaveCSSheet.
- + new even OnTextObjectMovePosition (move event) - OnTextObjectMove is still used for resize (unchanged)
- * several improvements in editor
- fix problem with Wordwrap and centered text

28.6.2009 -WPTools 6.04

- + WPTools Premium: Column Balancing



- * many improvements in RTF reader. Word documents are now understood better
- * Improvement in check for protected text (ppMergedText)
- + new ViewOptionsEx property
- auto hyperlinks were not working
- + TWPCoboBox has an event OnUpdateItems which will be triggered after the items had been automatically assigned.

24.6.2009 -WPTools 6.03.6

- * thinner page borders in thumbnail mode.
- ViewOptionsEx: wpAutoThumbnailMode will show pagenumbers only when in thubmbnail mode (= wpShowPageNRinGap in ViewOptions)
- + **property ColorDesktop and DeskGradientHorizontal to render the background with a gradient fill**
- * fix for protected text handling (CR after a field)
- * fix for text alignment near a movable image
- EditOption AutoDetectHyperlinks was not working anymore
- * WPreporter: SuperMerge.Stack.PageBreakAFTERGroup := true was ot working when footers were used

1.6.2009 -WPTools 6.03.5

- fix problem with display of character attributes when attributes were inherited from paragraph styles
- fix problems with selection deletion in single column, single row tables
- improvement of RTF writer when writing sections

11.5.2009 -WPTools 6.03.3

- improved report band dialog, new ShowOptions property
- fix in RTF reader to load header/footer
- change in HTML writer to save SPAN instead of FONT tag
- several fixes in editor
- * WPTools Premium: better column support. Fixed column height now splits correctly on 2 pages.

28.4.2009 -WPTools 6.03.2

- fix problem with justified text in PDF

21.4.2009 -WPTools 6.03.1

- fix problem with images when used in Delphi 2009
- better support for header/footer in RTF files created by word. (Ignore bogus header/footer)
- some stability fixes

25.3.2009 -WPTools 6.03

- + improved text rendering - optimation for character distances on screen to provide better display
- + improvement on ShowMergeFieldNames to improve cursor movement and drag and drop
- + automatic disable dragging of fields inside of fields
- + improved merge field selection. TextObject.SelectedObject now returns the mergefield if it was completely selected
- + change in HTML saving code to save src in after width adn height (for outlook)
- * various bugfixes

17.1.2009 -WPTools 6.01.5

- WPPREMIUM: Text after Columns initialized with WPAT_COLUMNS_Y is now allowed
- + TWPToolBar FontName drop down now lists fonts used by document first
- fix for tables which use a fixed row height and are splitted on different pages
- + improvements necessary for Delphi 2009 - the Localization demo now works
- + EditOptionEx wpDontPreserveObjectsAgainstDeletion
- fix problem in ImageObject LoadFromStream when GraphicEx is used
- fix problem with Delphi 2009 when loading WPreporter templates
- fix problem with HTML reader with paragraph style of first paragraph

26.10.2008 -WPTools 6.01

- * updated HTTP Demo, now with "Source View"
- + DELETE/BACKSPC at start of line removes right/center alignment
- + loads background images for paragraphs, tables and styles
- * improvement to text protection (empty lines)
- improvements to HTML and CSS reader
- improved HTML format routine
- improved MIME loading - now supports binary data despite Synapse does not)
- + MIME reader captures HTML body for SourceView
- * DataProvider now uses MergeText(",true) instead of MergeText
- + boolean wphhttp_Disable to disconnect HTTP temporarily
- * several changes to improve compatibility with Delphi 2009

17.10.2008 -WPTools 6.00.1

- several changes to fix problems which occurred with use of Delphi 2009
- * update to WPIO_MIME to also load binary encoded GIFS and JPEGs from EML files

13 Release Notes - WPTools 5

We include the WPTools 5 release notes here to provide You with hints to interesting changes and improvements.

14.12.2010 - V5.0 Service Release 58.1

- * workaround for MouseWheel UP being triggered too often (10 mms check)
- * some bog fiexe in editor, such as a possible problem when displaying symbols

23.7.2010 - V5.0 Service Release 58

- * several improvements of editor
- * improvement to RTF writer to when writing tabel cells
- * improved right aligned text
- * fixed problem with line heights of lines which are empty except for new line character

28.2.2010 - V5.0 Service Release 54

- fix problem in paint routine which causes problems when selection attributes

11.2.2010 - V5.0 Service Release 53.2

- fix in RTF reader to better load files which do not define codepage
- fix in paint routine to solve a rare lockup
- fix in format routine to improve section support

5.2.2010 - V5.0 Service Release 53.1

- fix of problem in save routine when footnotes were used (WPTools premium)
- fix in HTML writer

1.2.2010 - V5.0 Service Release 53

- fix problem with left aligned text and image wrap around (wrong alignment)
- fix problem with sometimes duplicated images in PDF export
- fix problem with black rectangle in first line under Windows 7, 64 bit

18.10.2009 V5.0 Service Release 51

- fix number style loading in WPT format
- some other fixes ported back from WPTools 7

31.6.2009 V5.0 Service Release 50'

- * Delete All Tabs in tab dialog really delete all tabs in selected text, not just the shared tabs
- * improvement of applying changes to selected text
- improvement of activation hyperlink hoovereffect
- + AppendAsSection now has optional Options parameter
- WPManageHeaderFooter.DFM was wrong version

28.6.2009 V5.0 Release 49

- * fix D2009 problem in Image support
- * other bug fixes

11.5.2009 V5.0 Release 48.3

- service release - fix problem with selections

29.4.2009 V5.0 Release 48.2

- fix problem with justified text in PDF

21.4.2009 V5.0 Release 48.1

- written merged cells more "Word friendly" to RTF
- updates to better support padding in cells
- some stability improvements

27.1.2009 V5.0 Release 47.1

- * improved code for better D2009 support (requires PRO)
- + publishes property FormatOptionEx2

17.1.2009 V5.0 Release 47

- + EditOptionEx wpDontPreserveObjectsAgainstDeletion - to avoid recreation of hyperlink objects after clearing selected text
- fix problem in ImageObject LoadFromStream when GraphicEx is used
- fix problem with Delphi 2009 when loading WReporter templates (requires PRO license)
- fix problem with HTML reader with paragraph style of first paragraph

25.11.2008 V5.0 Release 45

- * DataProvider now uses MergeText(",true) instead of MergeText
- improvement to load RTF correctle when using Delphi 2009
- fix to avoid AF due to wrong compiler optiomation
- fixed handling of buletted, centered paragraphs with tabstops

18.10.2008 V5.0 Release 44

- * WPTools PRO and PREMIUM:
 - several changes to fix problems which occured with use of Delphi 2009
- + new flag ProtectedProp: ppNoEditBeforeProtection
- no text insertion witin protected text at start of paragraph if prvios end of line was protected
- * ReplaceDialog will not replace protected text anymore

7.10.2008 V5.0 Release 43.2

- fix: Objects on other pages than first were not movable
- + FormatOptionEx flag: wpfDontIgnoreEmptyHeader to also keep distance for empty headers

29.9.2008 V5.0 Release 43.1

- some code fixes in HTML reader + writer
- WPManageHeaderFooter dialog focusses the main form so text input is possible after creating header/footer
- Code change to improve compatibility with DevExpress toolbar
- fix in setup exe

26.9.2008 V5.0 Release 43

- WPTools PRO / PREMIUM can be now compiled with Delphi 2009 (several changes)
- fix bug in InsertClass method
- GetPageOfBookmark works for all bookmarks
- Code to move images has been updated

24.9.2008 V5.0 Release 42

- * updated manual topic "Mailmerge and forms"
- ANSI writer: Separates cells by TABS and rows by CRNL
- DB Control now switches to readonly when dataset does not use AutoEdit

- Fix in preview dialog property DisabledDialogs
- PREMIUM: improvement to TextBox selecting
- improvement to MeasurePage event
- * some enhancements to API

10.8.2008 V5.0 Release 40.1

- * some minor bugfixes

15.7.2008 V5.0 Release 40

- * WPREMIUM: tables are handled in columns better
- * RTF did not save foreign characters in current codepage correctly when <> cp1252
- * WPREporter - a group with "always hidden" will never be processed
- * Better Landscape setting when printing - there was a problem with tables in sections

30.5.2008 V5.0 Release 39

- * small improvement to CSS reader
- some stability improvements
- PNG objects were sometimes not freed.
- fixed problem with EURO character

4.4.2008 V5.0 Release 38

- some small fixes in editor
- better text box positioning with WPPremium
- HTML reader - apply paragraph style to first paragraph as well
- fix when saving a single selected cell
- * Changed attribute detection for selected text - should work much better now.
This can be deactivated using const DefaultAttrAlsoForSelectedText = false in WPCtrRich but should be on for better editing possibilities.
- fix for missing bottom border before page break
- MoveToTable does not look for non-table paragraphs
- + RTFDataBlock.FindParByNameEx locates a paragraph with a given name and a given type

12.3.2008 V5.0 Release 37

- + LoadFromString procedure (more intuitive than property SelectionAsString)
- * apply bottom border when last row was deleted

22.1.2008 V5.0 Release 36

- fix of broken drag&drop support
- quicker deletion of text when clicking near text
- some fixes in rendering engine
- + detect PNG, GIF and JPEG automatically in TWPIImageObj.LoadFromStream (solves GIF saving problem)
- * improved Compress method in WPObj_Image.pas
- * change in WPIOHTML to not save empty header texts

20.12.2007 V5.0 Release 35

- fix to not draw border around table objects
- fix problem when pasting text in form completion mode
- fix problem is two DFM files
- fix problem with double grid lines (around tables)
- * improved update of GUI methods - now using inherited style properties (Use CUrrAttr)
- + When printing all colors can be set to black - see PrintParameter.PrintOptions: wpAllColorsAreBlack
- improved padding saving for table cells
- * the code dealing with reading attributes of the selected text now also reads attributes of attached styles.
This can be switched off using the EditOptionEx: wpDontInitSelTextAttrWithDefaultFont
If a character attribute was not defined in a style the document default is retrieved
This only works for character attributes, not for paragraph attributes. Those can be read using ActiveParagraph.AGetInherited in case SelectecTexAttr reports an undefined attributesx
- * Font "System" is now mapped to "Arial" since the cursor advance does not work otherwise

27.11.2007 V5.0 Release 34

- + WPRichText1.TextCursor.CurrAttribute.AInc(WPAT_CharFontSize, pt*100, 300) can now also be used to
 - increment/decrement font size of text which is controlled by a style sheet.
- * in normal layout modes borders are not drawn between virtual pages
- * Update to WPRReporter table calculation. When the functions left() and previous() refer to non-existing cell
 - The result of the function will be undefined and "[ERR]" will be displayed.
- * Some updates to RTF reader and HTML writer
- + ANSI text writer understand option "-nolinefeed" to use CR instead of CR+LF
- * some safety checks to avoid AVs wehn paragraphs are deleted
- + HTML reader loads <sup>
- * when in ppAllExceptForEditFields Mode the not editable merge fields will not be accessed by the TAB keys anymore. (can be switched off by compiler symbol MOVETO_ALL_FIELDS)
- fix in RTF reader for better import of text with several nested tables in one table cell
- + new format string for WPTools writer: -allnumberstyles, forces also the unused styles to be saved
- * WPRichText.Assign(Source) will also copy the number styles
- fix problem introduced by "fix AV when resizing table". Adding additional fix.
- + new flag ppIsHyperlink in property ProtectedPro
- fix of problem with wpShowInvisibleText
- + when pasting ANSI text (or one RTF line) the inserted text will inherit current paragraph attributes, except that
 - the new flag wpcoPastedANSIDoesNotInheritParAttr has been set in ClipbordOptions
- fix AV when resizing table an deleting a row at the same time
- fix problem with space_before and invisible merge fields start tags at beginning of line
- + function Print(PageRange): Boolean now also supports @@ODD@@ and @@EVEN@@ as page range short cuts

11.09.2007 V5.0 Release 30

- + SplitCells now includes a bollen parameter "before" (default=false)
- * EditHyperlink will insert the link as text if no linktext was provided in procedure call or as text selection
- + function WP.Printing to check if currently printing text
- * the TWPPreview will not paint itself while the attached editor is printing
- * ReformatAll(true, true) now clears all known character width (important for toggling visibility of fields!)
- * RTF reader loads \info\company and \info\manager + \info\hlinkbase
- + new method: SplitCellsVertically
- fix in ReplaceTokens. Tokens not seperated by spaces were not recognized
- * when pasting RTF text into an empty numbered paragraph the number property is retained.
- + new format string: IgnoreSpan for HTML reader
- + IgnoreSpan is automatically used for pasted HTML text (better for pasting from e-mails)
- fix in CodeLocate - compare ObjType

27.07.2007 V5.0 Release 25

- + Object (par or page relative) cannot be moved when wpobjLockedPos is used in the TWPTTextObj. Modes
- + TParagraph.IsUpperCase
- + TParagraph.IsLowercase
- fields with background color were displayed char by char which reduced display quality
- some small tweaks in engine

5.07.2007 V5.0 Release 24.4

- * changed virtual method DoUpdateEditState to DoUpdateEditStateEx
- + added event: OnEditStateChanged (replaces V4 EditChangeEvent)
- * image loading checks for empty stream
- fixed AV with ppNoEditAfterProtection

2.07.2007 V5.0 Release 24.3

- * updated OnToolBarIconSelection - removed "except end" when pressing "Open"
- some fixes to undo handling and tables
- some fixes to image handling (can be selected when outside page)
- * improved HTML reader (better handles non "X" HTML)

04.06.2007 V5.0 Release 24.1

- fix for dropped character problem (when first paragraph of all was longer than page)

- \sect which does not define section break in RTF code now inserts paragraph
- when deleting a table which was followed by another table UNDO will not insert the next table
- * paprlsHidden flag now saved to WPT format
- + new property: WP.RTFData.InsertTextIntoNewRow - copy in table always insert into new rows
- * EditBox mode checked after paste from clipboard
- + HTML reader loads UTF8 - use formatstring '-utf8'
- KeyPress does not check GetAsyncKeyState for space key
- selection was not removed sometimes on click
- fix in SetOuterCellBorders
- cell split could cause AV when UNDO was active
- pasting ignores merged cells
- + Support for Delphi 7 - Win32
- solves problem in Paragraph.Reformat when inserting tabs
- Shift+Delete is now handled (CutToClipboard)
- Ctrl+Left handling improved (problem with single char words)
- suppress bottom border in layout mode when outside of page area
- better handling of undo for page break added inside table
- 2 units in PRO version were wrong in V5.22
- WPWordDelimiterArray['-'] := true will disable word wrap in words with - sign
- paste ANSI was disabled
- DeleteFieldAtCP now works with protected fields, too
- end of page border line is not printed in normal layout mode
- fix in BulletStyleDlg - EditStyleNums
- don't position cursor in empty line under text box
- * change variable "new" to "newnr" to avoid C++ problem
- centered text with bullets now paints bullet besides the text, not at start of line
- WPPREMIUM - improved column reading/writing code (RTF format)

1.03.2007 V5.0 Release 23

- + WPPremium: Load&Save of column properties in RTF format
- + TParagraph.Exchange method to exchange two characters in the text
- * automatic hyperlink creation (wpAutoDetectHyperlinks) now moves trailing dots after the link
- table borders after long headers were painted wrongly
- when inserting a paragraph in a table cell the border attributes is not copied
- WPPremium: TextBoxes were sometimes not loaded at correct position
- WPPremium: Second Column was started one line higher
- WPPremium: Column break now starts a new column BEFORE the paragraph (expected behaviour)
- * updated Section reading from RTF (non breaking mode was checked too early)
- KeepN did now work correctly when space-after was defined
- + NumberStyles can now be loaded and saved (using GetWPCSS, SaveToFile)
- * updated unit WPIOWPTools: When loading numberstyles the ids are mapped and duplicates are removed.
- + FormatOptionsEx: new flag wpfNumberingControlledByStyles
If defined numbering modes are directly stored and used in paragraph styles.
Please note that this will only work when you use "WPT" format for load&save
- HTML writer created unnecessary open/close attribute tags before and after text objects

24.01.2007 V5.0 Release 22

- + improved auto indent code when numbering styles are used. (Can be switched off in EditOptionsEx wpDontUseNumberIndents)
- + alignment now updated for header/footer before paint code to provide proper alignment when fields such as "page number" was used.
Can be switched off using Memo. _DontUpdateObjInHeaderFooter := true
- + property RTFDataCollection.RTFViewOptions with flag wpLockDATEandTIME to lock the value of DATE and TIME fields
- improved drag&drop detection
- when RTF is loaded and text uses the charset 1 automatically the sysetm default code page is used
- * several improvements to HTML reader and writer
using the format string "-WriteAllCoWidths" all table cells will be written with a width param
"-DontWriteStyleParam" will switch off the saving of the inline styles style=""
"-csspath:..." can be used to specify a CSS style for loading and saving
- fix for line height problem when fields or bookmarks were used at the start of a line
- * several small stability improvements

26.11.2006 V5.0 Release 21.1

- + when saving HTML files embedded images will always be written as files in same directory as HTML file
- + new format option: -imgpath:"xxx" - embedded HTML images will be written to the path xxx
use -imgpath:"" to switch off saving of embedded HTML graphics.
- + HTML reader now detects ISO charsets
- improvement to RTF reader to better handle corrupt RTF files
- some improvements to engine
- fix to render engine to set font attribute of mailmerge tags (sometimes inherited attribute was used)
- better detection of black-on-black text
- * the optional ReportBuilder (9/10) support was rewritten and enhanced.
- + optionally FastReport support is now available

9.11.2006 V5.0 Release 21

- + new EditOptionEx: wpZoomWithMouseWheel to zoom with mouse wheel + Control
- invisible chars, such as bookmarks are now handled like spaces (ghost cursor bug)
- reactivated Ctrl+C - code was deleted by mistake
- some fixes to make upgrade from V4 easier (additions to finder, added functions GetSelTextBuf, InsertParText, ChangeAttr, GetParText) activated by compiler symbol V4COMAPT)
- removed reference to TransparentBitBlt in unit WPObj_image since not supported under Windows NT
- TWPTTextStyle.AGet_CSS has additional parameter "IgnoreMargins" to suppress the writing of margin and indent attribute (useful when writing tag
- * some improvements to HTML reader and writer
- * HTML reader will align tables in the middle by default (like IE)
This can be switched off by undefining HTML_CELLS_VALIGN_MIDDLE in WPIOHTML
- + new option for HTML reader: -onlyinbodytag, text outside <body> tags is always ignored.

15.10.2006 V5.0 Release 20.9

- + unless compiler symbol DONTREQ_SHIFT_FOR_UNDER_TEXT_OBJ has been set, images which are under the text
require the SHIFT key to be pressed to become selected
- * tabstops wrapped to next line were ignored (width=0) - now they will be used to jump to first tab.
(can be switched off in PRO version using compiler symbol DONT_USE_TAB_IN_NEXT_LINE)
- * improved response time after mouse click
- + the ANSI text reader will convert #12 chars (form feed code) into page breaks. (switch off with DONTUSEFF)
- changes to improve the usage of inherited font and font size information
- fix for line draw problem for table cells which span a page
- * several small changes to fix stability issues
- fix of problem in conversion of unicode to ANSI
- fix in HTML reader: alignment in DIV was applied to previous paragraph
- + load and save to unicode strings: GetUNICODE and SetUNICODE. (Internally the UNICODE reader/writer are used)
- + new: ruler properties IntervallStepsInch and IntervallStepsCM

11.8.2006 V5.0 Release 20.8

- + property WPRichText.Memo.TextColor - this color is used for the text which uses the default color.
It is initialized from the global variable wpCWindowText which has the default value clBlack.
Note: To set the color of the editor window use property DeskColor and PaperColor
- + new flags in ProtectedProp: ppBookmarkKeepStructure and ppInsertpointKeepStructure
--> if text is deleted the contained bookmark or field objects can be recreated.
- + new method: SetOuterCellBorders(Activate : Boolean; BorderWidth : Integer = -1);
It is now used by the "outer border" standard action
- * StartSpellcheck automatically positions cursor before current word.
- Some improvements to further improve stability
- + if compiler symbol TOTALREADONLY is active (see WPINC.INC) no table or object resizing is possible when the Readonly property is true
- + new ClipboardOption wpcoAlsoCopyHTML to also create a HTML block when copying text. This solves a
problem when the text is pasted into Outlook express which seem to not handle RTF correctly.
- * numbering action will now continue numbering after bulleted paragraphs
The numbering will restart, unless the CTRL key is pressed.

17.7.2006 V5.0 Release 20.7

- + new event: OnLeaveRTFDataBlock
- * improved HTML reader - assign border attributes from styles
- InputCode automatically moves to first cell if currently in table or table row object
- some security checks (GetPosition, SetPosition)
- + improved RTF reader (reading nested tables)
 - * fix for wrong headerr tag in WPWord written files
 - * ignore wrong textb code in TX written RTF files
 - * under some circumstances paragraphs were appended
- WPPREMIUM: KeepN was broken
- some stability issues

13.6.2006 V5.0 Release 20.6

- * don't color paragraph symbol when merge field at end of line
- fix in DeleteMarkeredChar API
- change in RTF reader to handle table rows with no cells
- added load formatstring option: -overwriteparattr to always use attributes of first paragraph
- + added REDO hot key Shift+Ctrl+Z (in addition to Ctrl+Y)
- * modified numbering action - <http://wpcubed.com/forum/viewtopic.php?t=2148> with changes to make it also work with selected text
- added similar code for bullet button
- * Table dialog now allows wpDefaultTableInTable if option wpAllowTableInTable was not set
- new: default Option wpNestingAsInEditOptions to use wpAllowCreateTableInTable of TWPRichText.
- EditOptions
- improvement of cursor position restore when applying UNDO

17.5.2006 V5.0 RELEASE 20.5

- + TIME and DATE text object now handle the data format string correctly
- + when using the tab key to navigate through a table the destination cell will be automatically selected unless you have specified wpDontSelectCellOnSpreadsheetMovement in EditOptionEx
- * The interface SelTextAttr will now report the default attributes if no other attributes are defined by the selected text. This behavior can be disabled with EditOptionEx wpDontInitSelTextAttrWithDefaultFont
- * Punctuation chars are now handled as 'words' when using Ctrl+Cursor left/right
- several fixes in engine
- * the save dialog can create a default extension if the extension does not match HTM, HTML, TXT, WPT, RTF or DOC
- (this must be enabled using compiler define CREATE_AUTOEXTENSION. By default the extension will be
- appended if it is just a number. Now the selected format (filterindex) will be passed to the save procedure.
- fix range check in cursor up/down (was introduced by XPosLineUpDown)
- better bullets in right aligned text
- StyleDialog hides TAB option unless compiler symbol STY_DONTHIDETAB was defined

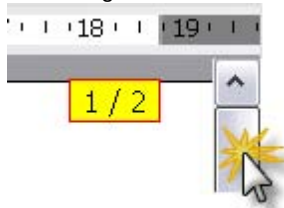
18.4.2006 V5.0 RELEASE 20.3

- + new powerful function FieldLocate which can be used to enumerate the fields in the document to read or write their contents. It was created to offer an alternative to Mergetext which does not require the use of a callback.
- * Cursor Up/Down now tries to retain the horizontal cursor position (can be switched of in EditOptionsEx)
- + MailMerge(name) now allows ONE wildcard character '*' in the name parameter
- + MailMergeEx(name, command) also compares the command property of the fields. Also allows wildcard
- + WPRichText.Memo.DisableBackgroundOnBWPrinter can be set to TRUE if (and only if!) there are problems with the printer printing colored text. Some old printers seem to "think" they should print the text using white color if the background style is clear. (doesn't make much sense)
- * other changes and stability fixes

01.3.2006 V5.0 RELEASE 20.2

- + new event PaintPageHint: Customize the canvas properties or paint the page number in your own code (then set "Ignore" to true)
- + new EditOptionsEx: wpAlwaysCoWidthPC when changing a column width all width will be calculated in %.

- + new global variables WPHTMLUL_ListImageURL_circle and WPHTMLUL_ListImageURL_bullet to set an image name for the HTML export to be used for items
- + CSS now understands the color 'transparent'
- + new options for Contents.Options in OnMailMergeGetText:
 - mmlIgnoreLoadedFonts and mmlIgnoreLoadedFontSize to ignore fonts when loading RTF
- some fixes in engine
- (!) fix in RTF reader to work when correctly when Czech list sorting is active
- + new event: OnTestForLinkEvent to detect hyperlinks in plain text.
- + new: [Ctrl+DELETE] deletes word or white space to the right
- + CSS format now supports the MS Office elements: mso-style-parent and mso-style-parent
- * better detection of symbol fonts
- * function IsSelected returns FALSE if text is selected but the length of the selection is 0 character
- + **new demo: PrintOnBMP** - for those who want to FAX pages
- + **new demo: PlainTextLinks** - have hyperlinks in plain text (without having <a> tags in text)
- + new procedure ScrollLinePos(par, posinpar) makes this text line first of screen (when text long enough)
- paste of tables in RTF code works better
- improved automatic field selection
- * better display of selection of paragraph breaks
- * also show manual pagebreak (dashed line) when property WordWrap=true
- + Display hint with "Page/ PageCount" when the text is scrolled. Can be switched off using compiler symbol NOPAGENHINT or ViewOption 'wpDontDisplayScrollPageHint' also see event: OnPaintPageHint



- + OnPaint event now allows flicker free painting
- several fixes to RTF reader (i.e. \line\par in header)
- fix for default tabstops in WordWrap mode
- + **New EditOption: wpSelectCompleteFieldAlsoWhenInside** - even wehn selection is done within a field the complete field will be selected.
- + **EditOption: wpDontSelectCompleteField** - unless activated, always select a complete fields when the selection contains part of a field
- fix for default tabstops in WordWrap mode
- + new ClipboardOption: wpcoDontCopyProtectedAttribute, dont copy the protected attr (RTF, WPTOOLS)
- * better support for the WordWrap mode (when page LayoutMode is selected)
- + **support for context menu key.** The event OnMouseDownWord will be triggered when the context menu key was pressed. If you use the new EditOptionEx: wpDontTriggerPopupInContextEvent the old behavior is established, the event will be then only triggered in OnMouseDown.
- * some fixes to improve stability

31.1 .2006 V5.0 RELEASE 19.7

- + the WPTools Version 7 Delphi setup now also includes the Package WPTools5_BCB10_W32 for the C++ Personality.
- It uses the pascal compiler symbol "NODB" since the linker error 'cannot find DB.OBJ' was persistent
- * Delphi 2006 files now compiled with D2006 - Update 1
- + DevExpressBars support (define USEEXPRESSBARS)
- + DoubleClick word selection now excludes ([]) signs. When clicked on this signs the matching character is found
- + ruler now shows page size and margins which were assigned in OnMeasurePage event
- better operation of rulers in normal layout mode
- better handling of tabs with justified text
- problem with spacebefore has been fixed (was sometimes used at start of page)
- RTF reader did not handle tables at the start of a file correctly (when importing from Word).

- + compiler symbol: NUMBERACTION_SIMPLE if defined the number button will create a simple list, not an outline!
- + for Version 4 compatibility: new methods SelectPages, PagesAsString
- * improved table header-row and footer-row handling.
- some additional checks to improve stability
- * improved RTF reader to better handle charsets

12.1.2006 V5.0 RELEASE 19.6

- + added support for Delphi 2006
- * character styles will not applied to the symbols of bulleted lines
- + negative tabs are now possible
- * the RTF reader will now apply the maximum column width to a table. That produces output as seen in Word. To switch it off globally use compiler define ALWAYS_IgnoreTableWidth or the reader format string "-IgnoreTableWidth"
- * The horizontal Ruler now allows it to move the right indent into the right page margin
- several improvements when reading RTF tables, better nested tables due to table start/end tags
- + Drag&Drop now supports auto scroll (move mouse close to, but not over the border)
- fixed problem with 'space_after'
- WPREporter: fix of editing bug in
- WPREporter: no par is appended after table when loading RTF text template (we suggest to use WPT)
- PreviewForm now updates page number in scroll event
- * if in SinglePageMode TWPPreview.PageCount now returns the PageCount in attached TWPRichText.
Also: Scrollbar will be updated according to pagecount if AutoZoom = fullpage
- + new FormatOptionsEx: wpfNoTableHeaderRows and wpfNoTableFooterRows to switch off the duplicate display of table header and footer rows.
- automatic decimal tabs in tables work better (word wrap bug) and automatically use ',' or '.' if the respective other char is not present

3.1.2006 V5.0 RELEASE 19.5

- InputEditField returns reference of *first* marker
- fixed problem with text in first cell of table which extends over left page margin
- * show empty "size" combobox whe selected text uses different font sizes (WPAAction.PAS)
- WPPREMIUM: fixed problem when a footnote was first char on a page

22.12.2005 V5.0 RELEASE 19.4

- fix for wrongly aligned images in header
- WPREporter copies tabstops
- RTFDataCollection.Clear did not clear numberstyles
- compiler define 'SAVE_ALL_NUMSTYLES_WPT' was set in unit WPIOWPTOOLS
- avoid one pixel padding for regular texts lines
- improvements to RTF reader
- AutoScrollFeature now disables itself when text cannot be scrolled any further
- * resizing of really small images now works
- + procedure PrintPages can now print in reverse order (to value < from value!)
- + support for double, tripple and quadro click to select word, sentence and paragraph
- + new EditOptionsEx wpDbClickCreateHeaderFooter
- + new event OnClickCreateHeaderFooter (create header/footer after double click in margins)
- + textobj.Name and txtobj.Source is now saved with image objects to RTF
- * some changes to make WPTools PRO and PREMIUM work with Delphi 2006
- * load black character background color from RTF

07.11.2005 V5.0 RELEASE 19

- + Ruler now have OnChanging and OnChanged events
- + If there is just one decimal tab in a table cell the text will be formatted as if it starts with a tabstop (can be switched off using FormatOptionEx wpfNoAutoDecTabInTable)
- * in paragraphs with borders the padding between paragraphs which use the same borderflags is now ignored
- * CodeInsideOf checks if cursor is on the closing object and then also returns the start object
- RTF reader had problems with WPREporter templates when groups were empty
- avoid wrap of last char when right tabstop is near right border
- fixed misplacement of cursor after first char in a wrapped paragraph (happened sometimes during typing)
- fixes in HTML reader/writer to better support tag

- WPTOOLS writer now writes the the tag <StandardFont/> correctly. Reader repairs previously saved tag.

28.10.2005 V5.0 RELEASE 18.10

- + faster initialization of the text (loading+pagination+display of test document "RTF Spec 1.8" = 220 pages, 8MB RTF in about 7 sec on 1.6 GHZ Test PC.)
- + faster typing in texts which very long paragraphs (1 par=40 pages!). This mode can be switched off in EditOptionsEx: wpDisableFastInitOnTyping - which should not be necessary.
- * change of 'DelayedInvalidate' logic in unit WPCTRMemo now using a timer. This avoids problems in MDI applications when a form was destroyed which uses a TWPPreview control
- MoveToField moved the cursor in EditFields one char to far.
- field 'NEXTPAGE' was not working
- when RTF was loaded merge fields removed the KeepN property from their paragraph
- fixed handling of the automatic tab to indent first (did only work when first indent was on left page margin)
- highlight of current field did not work (EditField demo)
- + FormatOptionEx: wpfKeepNUsesParImages - keepN also check for paragraph aligned images.
- + RTF reader/writer now supports the sbknone, sbkpage tags for sections
- * much improved performance when working with files which use SPAN styles
- * some improvements to paragraph border dialog
- Save section starts also before tables to RTF
- when loading sections saved with word not automatically page breaks are created
- fix small problem in HTML writer which caused </td> to be saved for merged columns
- Fix to KeepN support (only one block was supported per page)
- it was not possible to edit an 'undefined' value in the value editbox by typing. It stayed grayed.
- * some improvements to HTML reader/writer

17.10.2005 V5.0 RELEASE 18.8

- + new function GetImageAtXY - to locate all images in MouseMove event
- * revised UNICODE copy&paste
- * format routine now handles negative left/first indents
- + The format routine will *not* ignore empty paragraphs at the end of the footer.
If your Application needs this behaviour use the flag FormatOptionsEx :
wpfIgnoreTrailingEmptyParAtFooter
- fixed 'ghost image' problem
- fixed problem of center alignment of images in table cells when combined with new line characters
- fixed problem with section header/footer when section started with a table
- improvements to KeepN support. (note: KeepN requires 2 reformat runs)
- * ParProp dialog now shows inherited values in gray color

27.9.2005 V5.0 RELEASE 18.7

- * [WPReporter](#) will create long documents faster
- * in event OnTextObjGetTextEx the property TXTOBJECT.ParentRTFPage can no be used to know the page the object is painted. (Useful in header/footer)
- Paste in header/footer when displayed text is not the body does not create ghost RTFDataBlock ('DoubleVision bug')
- + new event: RTF.BeforeFormatTableRow to switch on KeepRowTogether for certain rows
- + new ClipboardOption wpcoDontCopyProtectedText - do not copy / cut protected text
- + [WPTools Premium](#) HTML export: footnotes with hyperlinks, textboxes as aligned tables

12.9.2005 V5.0 RELEASE 18.6

- * collection RTFProps.CharStyles is disabled - it was not used. Now ParStyles is consistently used for characterstylesheets, too
- * better support for character styles load&save in WPTOOLS format
- fixed format problem with space_between
- + RTF reader can now load RTF texts and use the existing style sheets with option "- dontoverwritestyles"
- improved handling of inc-indent action

6.9.2005 V5.0 RELEASE 18.5

- + new EditOptionsEx: wpAllowDrawDropBetweenTextBlocks - for Drag&Drop into/from header + footer texts
- + new AddCopy function in the paragraph style collection. Copys a style with all base styles from a different collection

- + WPreporter templates print much better since a left margin of 1/2inch is respected - group arrows are now visible.
- * improved nested tables: If a nested table is only cell content no padding will be used.
- bug fixes in RTF reader: Read styles with \pn group, nested tables
- WPreporter now works with paragraph styles even if Source/Dest does not use shared TWPRTFDataProps
- Save dialogs now works for TXT files
- fixed problems with NL in table cells and tabstops
- fixed problem with cursor positioning in right aligned paragraphs
- * better display of highlighted text (background color) with certain printer drivers and wPDF
- * TParagraph.CreateCopy now also copies style sheets which are used by this paragraph
- * images now belong to the TWPRTFDataCollection and not the TWPRTFProps. This avoids problems when
 - when several RTFDataCollections used the same RTFDataProps. Using the new FormatOptionsEx wpfStoreWPObjctsInRTFDataProps the TWPObjct can be still stored in the RTFProps - they will be only deleted if RTFDataProps.ClearAllWPObjcts is executed.
- * improved cursor movement, also in forms
- * WPTOOLS format now uses <StandardFont/> tag to save the current default font.
- + new procedure InsertRowAbove
- + when using the InsertRow action or toolbutton press CTRL to insert ABOVE of current row
- * TWPPreviewDlg now automatically uses events (watermarks) from attached editbox. (assigned using code: dia.WPPreView1.AssignPrintProperties(FEditBox);)

27.8.2005 V5.0 RELEASE 18.3

- RTF reader: unlock default font also in paragraphs which use a style
- fixed clipping problem (under rare circumstances the first line was not displayed)
- HTML reader/writer supports ID property (<p id="...">)
- + The HTML reader will create a page break after tag <pagebreak/> (useful for on the fly created HTML code)
- + The HTML reader will create a page reference to a certain bookmark with tag <pageref name="bookmarkname"/>
- + **Reference as HTML-Help (.cmh) file is now available** (see registered downloads)
- + CurrAttr Style, GetStyleEx, Color and BGColor now also report paragraph, paragraph style and Default Attr
- + FormatOptionsEx: wpDontAddExternalFontLeading - render lines smaller (more like WPTools 4)
- + new API: TWPOwnedCharAttrInterface.LockChanging, UnlockChanging - any attempts to change the attributes will be ignored.
- * **loading RTF will now modify the RTFData.ANSITextAttr to reflect the default font defined in the RTF.** (Use WPRichText.DefaultAttr.LockChanging to disable!) This is an important change since otherwise the standard font is always used.
- RTF-Reader: handle codepage in reader stack
- * disable drag and drop when multiple cells are selected (except for complete table!)
- * HTML reader: now allows , tags within <P> or <DIV> tags
- * Print() and PrintDialog functions automatically disable WordWrap property for the time of printing. You can define the symbol ALLOWWORDWRAPPRINT to disable this change
- padding-right in table cells was sometimes duplicated by format routine
- fixed paint problem with TWPRichTextLable which is using AutoZoom
- fixed problem with property ViewOptions: wpNoEndOfDocumentLine
- fixed problem 'clipdebug' not anymore \$defined in unit WPCtrMemo
- some small bug fixes in PreviewDlg + scrollbars

18.8.2005 V5.0 RELEASE 18.1

- + new IDE dialog (click right on TWPRichText control) to pre configure format- and edit options.
- * WPPREMIUM: InputFootnote now expects instead of the 'CreateNumber' boolean a new parameter a 'mode'
 - Using 'wpNumberInFootnoteIsSuperScript' the number in the created footnote will be super script.
- * REPORTBUILDER(tm) Support units have been updated, now with metafile cache for faster display
- * improved RTF style handling: all redundant character and paragraph attributes are removed automatically
 - (can be switched doff using FormatOption : "-DontFixAttr")
- + CurrAttr.FontName and Size will now report the default/style attributes.
 - Can be switched off using \$define DONT_REPORT_DEFAULT_ATTR
- + updated DefaultAttr handling. Using this property you can set the default font

- + new EditOptionsEx: wpDontResetPagesizeInNew - the "New" action will not reset the page size!
- + new EditOptionsEx: wpSetDefaultAttrInNew - the "New" action will reset the writing mode to the default
- + new method: ClearEx(DontClearStyles,DontResetPageSize,ResetWritingAttr : Boolean);
- * updated function Draw()
- * if EditOptionsEx flag 'wpClearAttrOnStyleChange' is used assigning a style will remove the redundant information first. This is not required with WPTOOLS format and RTF unless "-DontFixAttr" was used.
- + new property: AcceptFilesOptions. Create movable images or linked images with drag&drop!
- + new event: AfterLoadText - preprocess the text before after load operations
- + much improved Par.SplitCell method and new property EditOptionsEx wpAllowSplitOfCombinedCellsOnly (Disables a function which allows the creation of tables with different column count per row)
- PageProp dialog will not report custom size anymore
- fix of drag&drop of text and images between pages in multi column layout
- * RTF reader now appends an empty paragraph after tables. This can be disabled using the compiler define DONT_APPEND_PAR_AFTER_TABLES but should be better for the usability.
- + PrintParamameter.PageSides now works with the Print() function
- + The print dialog now also allows printing of selected text. This can be disabled using PrintOption 'wpDontAllowSelectionPrinting'
- + speed optimation of the RTF reader (especially for files with images)
- * removed the unused state element from record: TLine
- fixed spacing problem for paragraphs which were following a table
- some other small bug fixes and improved handling.

3.8.2005 V5.0 RELEASE 18

- * support for UNICODE copy&paste
- better support for UNDO in DeleteColumn
- * CombineColumns does not anymore combine cells in tables which are embedded in the selected cells
- + support for vertical alignment also in vertically merged cells
- + new property FormatOptionsEx
- + wpClearDoesNotDelete in the attribute 'locked' of paragraph styles and numbering styles is now working
- This property makes it easy to set up a standard set of styles which is available in new texts.
- * HTML reader/writer uses "width" element in style strings for images, tables and cells
- + new property: TWPStyleDlg.SaveCSSAsWPCSS. If true the WP-CSS format will be created when the user selects to save to CSS format. WPCSS supports all properties. WPCSS will also be saved when the file extension is wpcss.
- When the extension is INI or STY the INI format will be saved. This format also contains all properties. STY file created by WPTools 4 are now imported better.
- + new event: BeforePasteImage - possibility to change the embedded object.
- + new method: GetParXYBaselineScreen can be used to calculate the baseline of certain text, for example to set the position of a drop down menu.
- + TWPSelctedTextAttrInterface now supports: ToggleCharstyle
- Example: Implement Hotkey CTRL+B in OnKeyPress event:
if (Key=#2) then begin WPRichText1.TextCursor.CurrAttribute.ToggleCharstyle(WPSTY_BOLD); Key := #0; end;
- * changed property WriteObjectMode in 'default editor' to wobRTF. It was 'Standard'
- * improved TParagraph.SetStyle method which can remove properties which are defined in style
- several small bug fixes in RTF engine
- * Additions to this manual, see [Table Attributes](#), Mailmerge and forms, [TWPRTFDataCursor](#)

15.7.2005 V5.0 RELEASE 17.4

- + new PrintParamameter.PrintOption: wpAlwaysHideFieldmarkers - to hide field markers in Print; and PrintDialog;
- + new edit functionality: **when resizing a table while pressing CTRL key** the width of each column will be adjusted by preserving the aspect-ratio of the column width to the table width
- * editor improvements: better protection against unwanted change of column -width and -height.
- + improvements to HTML reader to display certain newsletter a lot better
- + WPRReporter: new property **ColumnWidthSnapValue** - make sure table columns use same width (the default value 15 maps column lines which are not further away than one screen pixel)
- + WPRReporter: new Option flag "wpFixAllColumnWidth" - converts variable width columns into fixed width columns
- + TOCs can now be also created from the first line in a table cell using the new mode flag [wptocAlsoProcessTables] for the CreateTableOfContents method.

- + WPAT_NoWrap can be used to switch off the word wrap in one paragraph.
This mode has to be specifically activated if property FormatOptionsEX
- * updated RTF load metafile routine
- * border dialog now displays state of current paragraph
- fix: draw paragraph border at end of page
- fix: additional hyphen drawn at end of line
- fix: Paragraph.MergeCell procedure improved

10.7.2005 V5.0 RELEASE 17.3

- + new localizable strings. See file "WPTools_EN_ADDED.XML" in Demos\Tasks\Localisation
 - + new procedure TWPOImage.Compress - automatically used after paste from clipboard.
Requires \$DEFINE COMPRESSBITMAPASJPEG
 - + RTF reader now supports vietnamese charset
 - + WPREPORTER: the option wppNewPageAfter can now be used for report groups
 - + WPREPORTER: new utility functions in WPSuperMerge:
FindGroup, FindBand: Locate bands and groups
 - ConvertLetterIntoTemplate:** Create a banded report template from a text with header/footer
(also see: ConvertTableIntoGroup: Convert a table into a report group with header, data and footer bands)
 - + [WPREPORTER: band.Bookmark](#) to automatically wrap the text which was created by a band or group into bookmarks. The bookmarks are applied after all text was copied!
 - + WPREPORTER: new event: AfterCopyParagraph - triggered after new par was appended but before the data is merged
 - + new component: TWPMergeHeaderFooterDlg opens the dialog to create and delete header/footer, now localizable!
 - * improved XMLEditor for WPLanguageControl
 - + new flag for Contents.Options in OnMailMergeGetText: mmDeleteThisField deletes the field markers!
 - + WPREPORTER: Option 'wpDeleteFieldsInDestination' deletes the fields in destination. (uses mmDeleteThisField)
 - + ReplaceAll through the replace dialog now
 - a) uses UNDO b) will work within current selection only
 - + Replacement in TParagraph.Replace now supports UNDO
- Bugfixes:
- Select Word Procedure
 - DB control automatically uses '-nobinary',
 - HTML reader and writer improvements
 - RTF reader now creates OnRequestHTTPIImage event of embedded TWPOObject (import from V4 files) and ignores next INCLUDEIMAGE
 - RTF reader: Ignore \r\n after unicode \u tag
 - RTF reader: apply subtractive properties
 - Fixes in WPTOOLS reader
 - some improvements to format and paint routine
 - + new event: AfterCopyToClipboard makes it possible to add custom objects to clipboard
 - + CombineCells now removes empty paragraphs
 - inputbuffer for fast writeres (unit WOCTrMemo) optimized

14.6.2005 V5.0 RELEASE 17.2

- [** IMPORTANT **] The LayoutMode "playLayout" did not hide header/footer before (as intended and described in manual)
This has now been fixed. Header and footer will be hidden.
Please make sure you use __playFullLayout__ to also show header/footer
- automatic header/footer rows: improved alignment with other rows
 - * new property "DefaultNumberIndent" (must be changed in code) which sets the default indentation for bullets and numbers applied by bullet dialog and bullet button
 - page numbers are grayed in header/footer
 - * the HTML writer can now create simple UL and OL lists. This feature can be deactivated defining the compiler symbol DONT_WRITE_SIMPLE_LISTS
 - + new event: OnCalcPageNr makes it easy to change the displayed page nr.
 - some improvements of formatting function for merged table rows
 - improvement of paint procedure (selected text and background colors)
 - + new SelectedTextAttr.ClearAttrOverride
 - + new ViewOption: wpDrawHeaderFooterLines
 - + new FormatOption: wpNoMinimumCellPadding

- + new: TWPTextStyle.ADelAllDefinedIn
- fixed memory leak with the 'StoreComplete' undo object
- * Copy text from single table cell does not copy borders and table structure anymore
- * removed unused interfaces from RTFEngine
- * CountPages, CountLines, CountParagraphs now reports '1' for empty documents (better for GUI since that first line will be automatically created when the editor gets the focus)
- They count the body text if the cursor is not set to any other RTFDataBlock

30.5.2005 V5.0 RELEASE 17

- + added new 'categories' to online help
- + WPPREMIUM + WPREPORTER: You can now use text boxes in report templates (fields can be used)
- + WPPREMIUM + WPREPORTER: You can now use four notes in report templates (fields can also be used)
- * Create table button always disabled when in table (unless wpAllowCreateTableInTable is used in EditOptions)
- * WPREPORTER: the StartCode of a group is now processed before the event BeforeProcessGroup!
- + new FormatOption: wpDontAdjustFloatingImagePosition - do not modify the position of relative objects to keep on page
- + new FormatOption: wpDontAdjustFloatingImagePosition - do not modify x,y of floating images automatically to avoid that they are outside of page
- + new option for TParagraph.LoadFromStream: "wploadpar_UseWritingAttr". Now the current attributes will be applied to the loaded text.
- + new: TParagraph.LoadFromString - makes it easy to set formatted text in a paragraph. (RTF or HTML commands are understood!)
- fixed problem with WPreporter template editing when layout mode was set to normal
- improved case insensitive search in finder
- several improvements to editor handling

12.5.2005 V5.0 RELEASE 16

- IMPORTANT: fixed problem with tabs in ruler which only occurred with Delphi 5.
- * some important improvements to cursor handling with WPreporter
- + keep properties of images when pasting (or TextObj.Insert) image while other image is selected
- + several improvements to WPreporter and Report-Band Dialog ([see new demo!](#))
- * improved writing of stylesheets in RTF code
- * improved saving of stylesheets in RTF code
- * the procedure ParStyle.LoadFromFile now has a parameter 'Merge' to merge in the new styles
- * improved stylesheet dialog (show focus in Listbox)
- * paint selection-background for selected text objects which use OnPaint event
- * the TWPRichText now publishes the event OnPrepareImageforSaving (used by the TWPRTFDataCollection) to make it easier to use in applications which do not use the "MultiView" feature. This event can be used to preprocess an image before it is saved, for example save it as GIF/JPEG file and store the file name in the property TextObj.Source
- + new function: TParagraph.InsertEx - makes it possible to insert text with \r signs (new pars will be created)
- This is internally used by TextObj.EmbeddedText := 'newtext' - so multiline text in bookmarks can be replaced
- + FORMTEXT fields are now converted to wptools edit fields
- + changed behaviour of FormatOption: wpfHangingIndentWithTab - the left indent will be handled as first tabstop not only if the tab is the first character but also if the text before the tab fits into the first indent.
- * OnTextObjectMouseUp, OnTextObjectMouseMove and OnTextObjectMouseDown are now also triggered for non-image TWPTextObj. (See editBox demo for a checkbox example)
- Only with image objects the x and y parameters are relative to the objects coordinates!

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) applies a string which was created by GetWPCSS
- * protected text is now also locked for attribute changes (unless ppDontProtectAttributes is used in ProtectedProp)

- * 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!
- + new viewoption: wpDrawPageMarginLines to draw a dotted line round the text area
- + function 'CodeListTags', works like 'GetCodeTags' but creates a TWPTTextObjList
- * 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
- * improvement to HTML writer for better tables
- + 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: cursor movement within ootnote blocks using cursor up/down
- + "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](#))
- * 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,empty) has been updated
- + 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 allows it to change the left indent only
- * better update of WPComboBox
- * 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: "-nomergetexts" - 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 performance and reliability

V5.0 - Release 13 (11.3.2005)

- solved problem which occurred under windows 98 (critical fix)
- + new IDE context menu for the TWPRichText 'Change Page Size'
- + 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. Please see the new demo "[SubParagraphs](#)"!

- * InputTextField/InputTextFieldName are now functions returning the created TWPTTextObj
- + 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](#)

V5.0 - 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
- * better handling of merged rows - now the last column can be always merged even if the column count in the rows is different
- + option: "-nobinary" to save RTF code with hex encoded data
- + save binary RTF variables
- * 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.

V5.0 - 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 separated by empty lines
- + PrinterParameter are working now. Please check out the **new demo "PrinterSet"**
- * updated chapter 'Mailmerge'
- * new chapter [TWPTTextObj 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

V5.0 - 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 Version 7. (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
- + 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
- * LoadFromFile now uses the FormatString parameter

V5.0 - 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 TWPTTextObj and Mode parameter to adjust painting for different devices
- several bugfixes (see history.txt)

V5.0 - 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 with 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 programming API

V5.0 - 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, multiple 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
- + WPreporter_Calc - requires WPTools bundle: powerful calculation in text and tables

V5.0 - 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)

V5.0 - 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

V5.0 - 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

V5.0 - 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

V5.0 - Release 6.5 (4.9.2004)

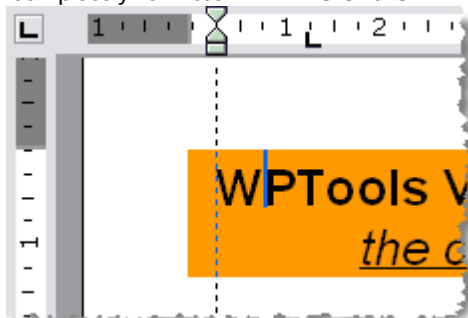
- + completely rewritten **TWPRuler**

(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

V5.0 - 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
- * 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](#))
- * WTools 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

V5.0 - 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 improvements to HTML loading and saving, especially 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)

V5.0 - 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 OnDbClick. The last parameter 'Ignore' now is passed as 'var'

V5.0 - 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 and tags
- improved HTML writing: saving of
 tag
- * added OnChange event
- removed LayoutModes 'wplayFullLayoutColumns' and 'wpThumbNailViewNr' (redundant)

V5.0 - Release 4.5 (25.7.2004)

- improved handling of soft line breaks = Char(10)
- improved action handling
- new pseudo action: 'TWPToolsCustomEditContoAction' which is used to replace the TWPToolControl
- added TBX demo (also see Use WPTools5 with TBX)

V5.0 - 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

V5.0 - 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)

V5.0 - 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.

V5.0 - 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.

14 PDF Products

WPCubed GmbH also markets this PDF products:

wPDF V3

Powerful and versatile PDF creator for Delphi and C++Builder.

It supports WPTools (see "[PDF export with wPDF](#)") and the most important Delphi reporting engines. Also converts metafiles into PDF data.

New Features in wPDF Version 3

- Support for standard brush styles (hatching)
- TransparentBitBlt
- Automatic reuse of the same image data. This way, when you export a document which often uses a logo the PDF file will be significantly smaller!
- Creation of [PDF/A](#) compliant PDF files (with added meta data and PDF tagging when you use it with WPTools)

- Support of [CID fonts](#) (known as "unicode" support)
- Binary Data embedding: You can store the document source which was used to create a PDF file within this PDF file. When you use WPTools for the PDF creation you can store the RTF source document inside of the created PDF data, the user only has to click on an icon to extract this document. This can be a great feature if you use it to embed programming examples within your programming manual!

Code as simple as this will export the text in PDF format:

```
uses ..., WPPDFWP, WPRTEDefs, WPCTRMemo, WPCTRRich;

procedure TForm1.ExportFromWPTools(Sender: TObject);
var pdf : TWPPDFExport;
begin
  pdf := TWPPDFExport.Create(nil);
  pdf.Source := WPRichText1;
  try
    pdf.FileName := 'c:\exported_rtf.pdf';
    pdf.Print;
  finally
    pdf.Free;
  end;
end;
```

WPViewPDF Version 3

PDF view, -print and manipulation technology by WPCubed GmbH

WPViewPDF is a component to load one or many PDF files to display or print as one. It is possible to export pages as bitmaps or as text. It is possible to add drawings which will be displayed and printed on top of the original data. It is possible to change field data, for example to fill out forms.

With WPViewPDF PLUS you can also add graphical objects and images to the PDF data (stamp PDF). It is possible to combine several PDF files into one new (merge PDF). It is also possible to save selected pages (extract pages) or delete certain pages.

Welcome to the exciting new Version 3

The Version 3 is the result of extensive work. We completely re-thought the logic which is required to load, render and manipulate PDF data to create this new version. It makes use of clever and effective caching for quick response times. It also makes use of multithreading for better user interaction.

We revised the PrintHDC method - printing to any windows device should be now much easier to do than before and produce higher quality.

The multithreaded scrolling viewer can quickly change between zoom states and various layout modes, including multi column display and side by

side page layout. It can also display a separate thumbnail view to the PDF.

Unlike version 1 and 2 the new version 3 uses floating point numbers for graphic output which offers better print results for many PDF files. Despite the higher text rendering quality, printing will be faster since less data has to be transferred to the printer.

Version 3 PLUS introduce a new stamping method which also it to place objects or highlighting rectangles on the page. This objects can be moved and sized by the user. But we also implemented the scripted stamping because it makes it so easy to add titles or page numbers to a range of pages.

Text extraction now also creates text in rich text format (RTF) - here the logic tries to make use of PDF tags to keep text together which belongs together.

The field support has been enhanced for better compatibility with existing PDF files. We work to add the ability to *create* new fields to the "PLUS" Edition.

Why do I need a PDF viewer component?

- If you need to embed a PDF viewer into your application, then you need WPViewPDF since this will, most likely, no longer be allowed with the Acrobat(tm) Viewer Version 6 or later.)
- If you need to load PDF files from memory, then you need WPViewPDF which will allow you to load PDF files from any stream. The stream interface makes it possible for you to use your own encryption/decryption scheme for the loading process.
- If you need to print the PDF files created by your own application, then you need WPViewPDF which makes it possible to print several PDF files using just one printer job without starting any external application
- If you need to use information from PDF files as background images in your application, then you need WPViewPDF since it has the ability to extract PDF pages as metafiles or print to a windows device (HDC).
- You can offer the user the ability to add custom texts and highlighting areas to a PDF file.
- You can extract text from PDF under program control
- Create a transparent highlight rectangle on a page and move it under program control (or let the user drag and move it)
- Read and write (PLUS Edition) to fields on PDF forms. This makes it possible to fill out such forms under program control.
- Last but not least: Imagine a powerful and versatile print and preview which is based completely on PDF files. The PDF files can be viewed, printed (with WPViewPDF or Acrobat(tm) Reader), stored or send via e-mail!

History of WPViewPDF

WPViewPDF V1 was created in 2003, mainly as viewer for PDF files which were created by our own PDF engine. While version 2 already produced much better display than V1 it still suffered from the limitation to internally use a

graphics library which was based on integer coordinates.
So we decided to rewrite most of the code for version 3.

Index

- A -

Actions 32
Animated GIF 184
Attributes 115, 120
AutomaticTextAttr 51

- B -

Bookmarks 196

- C -

C++Builder 272
Custom Draw 154, 174, 191

- D -

Default Editor 95

- F -

FastAppendText 36
FieldLocate 59
Format Strings 240
Forms 61

- G -

GIF 184

- H -

HTML 184
Hyperlinks 42

- I -

InsertPointAttr 51

- L -

LabelPrinting 126
Language 83
License 10
Load & Save 240
LoadImageFromFile 54
LoadImageFromStream 54
Localization 83

- M -

Merge Images 54
Multithreading 66

- O -

OnMailMergeGetText 53

- P -

PDF Export 198, 311
PDF View 311
Printing 72, 159, 163, 164

- R -

RTF2PDF 1

- S -

Sections 167
ShowMergeFieldNames 51
Splitscreen 149

- T -

Tables 112, 115, 145
TextDynamic 1
Token Conversion 226
TParagraph 14
Translation 83
TWPRTFDataBlock 14

- W -

Watermarks	154
Webbrowser	184
wPDF	198
WYSIWYG	74