DataStructures of Word Processing Tools

See also Example

unit WPDefs, WPTxtDef

Please note that these definitions are subject to alterations and modifications pending further development of the program!

The following classes:

TWPRTFText
  TWPRTFTextIO
    TWPRTFTextPaint
      TWPRTFTextInp

have the pointer

FirstPar : PTParagraph

which points to the first paragraph:

```
  PTParagraph = ^TParagraph;
  TParagraph   = record
    indentfirst, indentleft, indentright   : Integer;
    spacebefore, spaceafter, spacebetween : Integer;
    align             :TParAlign;
    prop              :TParProp;
    state             :TParState;
    Border            :TBorder;
    Tabs              :Longint;
    line              :PTLine;
    next,prev         :PTParagraph;
    end;

  PTLine = ^TLine;
  TLine = record
    Height   : Word;
    Base     : Word;
    y_start : LongInt;
    state         : TLinState;
    block         : (No, NoInverted, YesInverted, Yes, UnDef);
    pc            : Pchar;
    pa            : PTAttr;
    pmax,plen     : Word;
    lasttab, spacewid, spacerest : Word;
    next,prev     : PTLine;
    end;
```

The Structure looks like ...

```
  TLinState   = Set of (listMustPaint,listAutoNewPage,listHyphenate);
```

```
TborderType= set of (BlEnabled,BlFinish,BlTop,BLBottom,
    BlLeft,BlRight,BLBox,BLDouble,BLDot,BLHair,BLBar,
          BlInnerEdge,BlOuterEdge,BlCenter,BlTabLines);
 PTBorder= ^TBorder;
 TBorder = record
      LineType : TBorderType;
      Thickness : Byte;
      Color : Byte;
      { LO nibble = Color 0..15, HI Nibble = space in mm   }
 end;

 TParProp     = Set of                    (paprNewPage,paprFrame,paprBullett);
TParState   = Set of                (pastMustReformat,pastInitialize);
TParAlign = (paralLeft, paralRight, paralCenter,                paralBlock);
```

Attributes

Structure to show how to access the data in a TWPRichText control

```
WPRichText1.Memo.FirstPar
--> TParagraph.prev = nil
    TParagraph .line
    --> TLine.prev = nil
        TLine.pc -> 'Text'
        TLine.pa -> Attributes (TAttr)
        TLine.next
        --> TLine.prev
            ....
    TParagraph .next
--> TParagraph.prev
    ...
```

Word Processing Tools Version 1.5.5

WordProcessing Tools for Delphi

License Agreement

WPTools was developed by Ziersch, Munich, Germany.

Release date of this document: 4/1/1996 (Version 1.51)

email: 100744.2101@compuserve.com

Word Processing Tools

LICENSE AGREEMENT

WITH THE REGISTRATION YOU RECEIVE THE RIGHT TO DISTRIBUTE THE PROGRAMS YOU CREATED USING WPTOOLS. YOU MAY NOT DISTRIBUTE MODULES WHICH MAY BE USED BY A THIRD PARTY (VBX,VCL,DLL).

WITH REGISTRATION YOU RECEIVE ALL SOURCE FILES. THEY MAY BE ALTERED, BUT YOU CANNOT DISTRIBUTE THEM TO ANY OTHER PERSON WHO HAS NOT REGISTERED!

ALTHOUGH I SPENT A LONG TIME DEBUGGING THIS VCL THE FOLLOWING AGREEMENT IS NECESSARY FOR LIABILITY REASONS:

THIS 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.

IN NO EVENT SHALL JULIAN ZIERSCH BE LIABLE FOR DAMAGE OF ANY KIND, LOSS OF DATA, LOSS OF PROFITS, BUSINESS INTERRUPTION OR OTHER PECUNIARY LOSS ARISING DIRECTLY OR INDIRECTLY. 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 noncritical data before they can be relied upon.

record TAttr

See also

unit WPDefs;

Attention! Definitions will be modified (and still compatible) for future component versions.
Only footnotes and objects are currently (3/28/1996) planned.

```
WrtStyle=Set of (
          afsBold,
          afsItalic,
          afsUnderline,
          afsStrikeOut,
          afsProtected,
          afsHidden,
          afsHyperLink,
          afsIsMarked,
          afsIsObject,
          afsIsFootnote
          );

TDataType = (dtChar, dtObject, dtFootnote);

PTTextObj = ^TTextObj;
TTextObj = class
     Name          : string;
     tag           : Longint;
     prev, next : PTTextObj;
     DataType     : (dtGraphic, dtComponent, dtCalculation);
     data          : TObject;
  end;

PTFootnote = ^TFootnote;
TFootnote   = class
     Typ   : (fnChar, fnNumber);
     c      : Char;
     num   : Integer;
     text : String;
     prev,next : PTFootnote;
  end;



TAttr = record
     Width          : Word;
     Height         : Word;
     Style          : WrtStyle;
     case TDataType of
       dtChar      : (Base     : Word;
                      Font     : Byte;
                      Color    : Byte;
                      Size     : Byte;);
       dtObject   : (Data     : TTextObj);
```

```
   dtFootnote: (Footnote : TFootnote);
end;
```

DataStructures

TDBWPRichText

See also Properties Methods Events

Declaration : TDBWPRichText = class(TWPCustomRichText)

TDBWPRichText allows you to edit formatted text of almost any length. Thus you finally have the possibility to store huge texts in your database. The text must be stored in a Memo or in a BLOB field. TDBWPRichText can load RTF or normal ANSI format and is able to switch formats automatically.

TWPRichText

HyperLinkCursor
NoBlockMarking
OneClickHyperlink
MemoryFormat
CaretDisabled
FitToWindowHorz
SaveFormat
LoadFormat
Zooming
Resizing
BackgroundColor
Runtime only:
SelText,SelLength,SelStart
LastError
Lines
Modified
CPLineNr
CPLine
CPPosition
IsLastLine
SetModeControl

FontSelect
SaveAs Save
Load Insert
PrinterSetup Print!mPrint Print_xywh
ReplaceDialog FindDialog
DeleteSelection
PasteFromClipboard + Copy/CutToClipboard
LoadFromStream LoadFromStreamWithReader
SaveToStream SaveSelectionToStream
SaveToStreamWithWriter SaveSelectionToStreamWithWriter
SaveToFile LoadFromFile
Clear
LoadTemplate
WriteStatus
Find
FindNext
Spell_...
GetFontNr
GetFontName
Set_PageSize
Set_ParSpacing
Set_ParMargin
Change_ParSpacing
Set_ParBorderFinish
Set_ParBorder
PutParText,InsertParText,GetParText
InputText
GetTextLen GetTextBuf
SetSelTextBuf SetTextBuf
SetPosition

HyperLinkEvent
PrintFooter
PrintHeader
EditStateEvent
StartSpellCheck
CtrlTabPressed
ShiftTabPressed

TWPAltStatusBar

See also Properties Methods Events

Declaration : TWPAltStatusBar = class(TWPStatusBarBasic)

In contrast to the WPToolBar and the WPRuler the WPAltStatusBar is suitable for all your projects! The WPAltStatusBar differs from the WPStatusBar in design only.
You can define a set of strings which can be easily accessed from the program. You can also use a guage which easily deals with re-occuring functions. In addition, you can form a type of button from every string without using a separate handle for it. And, last but not least, WPStatusBar orders the strings when you re-size the window.

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg
Gauge Control

Strings
RtfEdit
ShowSizer
ShowGauge
GaugeValue
GaugeWidth
UseGaugeForWP

GaugeRestrictSub
GaugeIn
GaugeOut
GaugeInc
GaugeReset
SetStringStyle
SetStringStyleIndex
SetString
SetStringIndex
SetStringWidth
SetStringWidthIndex

OnSelection
OnUpdate

TWPMemoryFormat

See also

Declaration : TWPMemoryFormat = (fmRichText, fmPlainText);

TWPMemoryFormat represents the way how wptools holds the data in memory.

fmRichText will store for each character additional 10 bytes for the attributes.

WPRichText
DataStructures
TAttr

TWPPagePropDlg

See also Properties Example

Declaration : TWPPagePropDlg = class(TComponent)

WPPagePropDlg is a VCL which facilitates adjustment of the page size. You only have to set the property 'EditBox'. Then you can start the modal dialogue with Execute. The VCL uses the procedure WPRichText.Set_PageSize to set the page size, too.

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg

EditBox

```
procedure TRtfTextEditForm.PageSize1Click(Sender: TObject);
begin
   WPPagePropDlg1.Execute;
end;
```

TWPParagraphBorderDlg

See also Properties Example

Declaration : TWPParagraphBorderDlg = class(TComponent

This component makes it easy for you to set the respective border attributes. The attributes will be set for the actual paragraph or the selected paragraphs.

You may choose a color and the line width.
If you define a space (in mm) then the lines will be drawn at the margins, but the text itself will be indented.

The component uses the procedure TWPRichText.Set_ParBorder(typ, width, color, space) to define the properties.

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg
Set_ParBorderFinish
Set_ParBorder

EditBox

```
procedure TRtfTextEditForm.ParagraphBorders1Click(Sender: TObject);
begin
  WPParagraphBorderDlg1.Execute;
end;
```

TWPParagraphPropDlg

See also Properties Example

Declaration : TWPParagraphPropDlg = class(TComponent)

This VCL lets you define the margins (indentation) and spacing.The new values will be applied to the actual or the selected paragraphs. To use the component you only have to set the property EditBox.

The Component uses TWPRichText.Set_ParSpacing(before, between, after) to define spacing. TWPRichText.Set_ParMargin(first, left, right) is used to set the margins.

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg

EditBox

```
procedure TRtfTextEditForm.Indentation1Click(Sender: TObject);
begin
  WPParagraphPropDlg1.Execute;
end;
```

TWPRichText

See also Properties Methods Events

Declaration : TWPRichText = class(TWPCustomRtfEdit)

WPRichText allows you to edit formatted text of almost any length. You can choose to reduce the text attributes in the saved version, thus allowing you to process re-formatted texts larger than 2 megabytes.

When designing your program you can input text which contains various formats. WPRichText supports alignment, paragraph spacing, borders, font selection, colors, etc...

To automatically input text by your program you can use the procedures PutParText or InsertParText and InputText. You can assign the property lines the strings of a memo or listbox. You can assign the text stored in a TWPRtfStorage component to RtfText.

ToolBar
DBWPRichText
TWPRichTextLabel

HyperLinkCursor
NoBlockMarking
OneClickHyperlink
MemoryFormat
CaretDisabled
FitToWindowHorz
SaveFormat
LoadFormat
Zooming
Resizing
BackgroundColor
RtfText
Runtime only:
SelText,SelLength,SelStart
LastError
Lines
Modified
CPLineNr
CPLine
CPPosition
IsLastLine
SetModeControl

FontSelect
SaveAs Save
Load Insert
PrinterSetup Print!mPrint Print_xywh
ReplaceDialog FindDialog
DeleteSelection
PasteFromClipboard + Copy/CutToClipboard
LoadFromStream LoadFromStreamWithReader
SaveToStream SaveSelectionToStream
SaveToStreamWithWriter SaveSelectionToStreamWithWriter
SaveToFile LoadFromFile
Clear
LoadTemplate
WriteStatus
Find
FindNext
Spell_...
GetFontNr
GetFontName
Set_PageSize
Set_ParSpacing
Set_ParMargin
Change_ParSpacing
Set_ParBorderFinish
Set_ParBorder
PutParText,InsertParText,GetParText
InputText
GetTextLen GetTextBuf
SetSelTextBuf SetTextBuf
SetPosition

HyperLinkEvent
PrintFooter
PrintHeader
EditStateEvent
StartSpellCheck
CtrlTabPressed
ShiftTabPressed

TWPRichTextLabel

See also Properties Methods Events

Declaration : TWPRichTextLabel = class(TWPCustomRtfLabel)

WPRichTextLabel is a component which displays formatted text which cannot be scrolled or edited. Instead the text can be made transparent. Thus WPRichTextLabel is a efficient and powerful tool to create impressive presentations.

WPRichTextLabel originates from TGraphicControl. In contrast WPRichText stems from TCustomControl. This means that it uses the Canvas and WindowHandle of its Parent object.

WPRichText

HyperLinkCursor
NoBlockMarking
OneClickHyperlink
FitToWindowHorz
RtfText
Runtime only:
CPLineNr
CPLine
CPPosition
IsLastLine
SaveFormat
LoadFormat
Zooming
Resizing
BackgroundColor

GetFontNr
GetFontName
Set_PageSize
Set_ParSpacing
Set_ParMargin
Change_ParSpacing
Set_ParBorderFinish
Set_ParBorder
GetTextLen mGetTextBuf
SetSelTextBuf SetTextBuf
SetPosition

HyperLinkEvent

TWPRtfStorage

Properties Methods Example

Declaration : TWPRtfStorage = class(TComponent)

unit WPRtfIO;

TWPRtfStorage is a Component to store formatted Text (usually during designing) and assign it to any visual RTF component, if desired.

TWPRtfStorage has only one property:
```
 property   RtfText;
```

and only a few methods

```
procedure    Clear;
function     LoadFromStream(s : TStream): TWPLoadFormat; virtual;
procedure    LoadFromStreamWithReader(s : TStream;Reader:TWPTextReader);
```

RtfText

LoadFromStream
LoadFromStreamWithReader

```
WPRichTextLabel1.ButtonClick(Sender:TObject);
begin
   WPRichTextLabel1.Visible := FALSE;
   WPRichTextLabel1.RtfText.Assign(WPRtfStorage1.RtfText);
   WPRichTextLabel1.Visible := TRUE;
end;
```

TWPRuler

Properties Events

Declaration : TWPRuler = class(TCustomControl)

WPRuler represents the ruler in WPRichText and is completely managed by it as soon as the respective property is selected in WPRichText.

The only property which is important for the user is "units". You can select either `Centimeter' or `Inch' even when the program is running.

One Ruler can handle several WPRichText components, even if these are not located directly under one another. The ruler is simply moved accordingly.

Units
XOffset
ShowAllTabPos

OnSelection
WhileSelection

TWPSpellCheckDlg

See also Properties Example

Declaration : TWPSpellCheckDlg = class(TComponent)

This is an example component which shows how to implement spellcheck for a WPRichText or DBWPRichText component. You will find the source code in the unit Wpspdlg1.PAS. Since TWPSpellCheckDlg does not contain a dictionary, this is only an example.

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg

EditBox

This is a part of the sourcecode:

```
procedure TWPSpDialog.IgnoreClick(Sender: TObject);
var
 s : String;
begin
 while TRUE do
 begin
  EditBox.Spell_FromCursorPos;
  s := EditBox.Spell_GetNextWord;
  if s='' then break;
  { test Dictionary .... ##################################
    if not InDictionary(s) then
    begin
       EditBox.Spell_SelectWord;
       WordList.Strings.Assign( GuessList );
       break;
    end
    else continue;

##########################################################
######## }
   { without a dictonary any word is unknown: }
      EditBox.Spell_SelectWord;
      break;
 end;
 Word.Text      := s;
 ReplaceAs.Text := s;
 if (s='') and (fsFirstCall=FALSE) then close;
end;

{ Replace will replace the selected Text.
  If you use the procedure
  EditBox.Spell_ReplaceWord(s : string) the replacment will be done
  invisibly.
  If you have a non modal spell dialog, the use of Spell_ReplaceWord might
  cause errors if the user has changed the text meanwhile.
}

procedure TWPSpDialog.ReplaceClick(Sender: TObject);
begin
  if CompareStr(EditBox.SelText,Word.Text)=0 then  { Replace selceted Text }
     EditBox.SelText := ReplaceAs.Text;
  IgnoreClick(Sender);
end;
```

```
procedure TWPSpDialog.CancelClick(Sender: TObject);
begin
  EditBox.Spell_FromCursorPos;
  Close;
end;
```

TWPStatusBar

See also Properties Methods Events

Declaration : TWPStatusBar = class(TWPStatusBarBasic)

In contrast to the WPToolBar and the WPRuler the WPAltStatusBar is suitable for all your projects! The WPAltStatusBar differs from the WPStatusBar in design only.
You can define a set of strings which can be easily accessed from the program. You can also use a guage which easily deals with re-occuring functions. In addition, you can form a type of button from every string without using a separate handle for it. And, last but not least, WPStatusBar orders the strings when you re-size the window.

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg
Gauge Control

Strings
RtfEdit
ShowSizer
ShowGauge
GaugeValue
GaugeWidth
UseGaugeForWP

GaugeRestrictSub
GaugeIn
GaugeOut
GaugeInc
GaugeReset
SetStringStyle
SetStringStyleIndex
SetString
SetStringIndex
SetStringWidth
SetStringWidthIndex

OnSelection
OnUpdate

TWPToolBar

See also Properties Methods Events

Declaration : TWPToolBar = class(TCustomPanel)

The ToolBar serves to control one or more TWPRichText or DBWPRichText components. For this purpose the toolbar makes pre-defined speedButtons and Listboxes available. This makes choosing the font type or size, or even the text color, much easier. SpeedButtons include swithches for text styles such as 'bold' or alignment 'justified'. If you wish to link up other objects you can easily do so with AddControl. The ToolBar will then order the objects added together with the pre-defined objects. Most events which occur when an element is selected in the ToolBar are dealt with in WPRichText-DBWPRichText. The command is sent to the edit field which was last selected.
However, you may want the program to react differently. Then you have to prepare a Handler for the OnSelection and OnIconSelection event. This is especially necessary if you want to 'select' the New and Close Button. If you don't have a WPRichText or a DBWPRichText object in your interface window the ToolBar can also change the font type for TMemo, TEdit or TPanel objects. Just list the name and/or the object in the property UpdateObjectName/UpdateObject.

If the ToolBar is set for 'aligntop' or 'alignbottom' the heigth of the ToolBar is modified according to the space needs in the components.

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg

UpdateObject
UpdateObjectName
ShowFont
sel_ListBoxes
sel_StatusIcons
sel_ActionIcons
sel_DatabaseIcons
sel_EditIcons
FontSizeFrom
FontSizeTo
Hints

GetIcon
GetStyleBox
GetStyleItems
GetFontBox
GetFontItems
GetSizeBox
GetSizeItems
AddControl
RemoveControl
SelectIcon
DeselectIcon
EnableIcon

OnSelection
OnIconSelection

CtrlTabPressed

Declaration : property CtrlTabPressed : TNotifyEvent;

You may change the focus within an eventhandler, which is called when the user pressed Ctrl+TAB This is useful if the property WantTabs is set to TRUE, otherwise the TAB Key can be used to jump between text objects.

EditStateEvent

Example

Declaration : property EditStateEvent;

If the contents of the clipboard have changed you'll get an EditStateEvent. You may use it to disable/enable menu items.

```
procedure TForm1.EditStateEvent(Sender: TObject;
  selection_marked, clipboard_not_empty: Boolean);
begin
  { Cut/copy/Paste-Item are of type TMenuItem }
  CutItem.Enabled := selection_marked;
  CopyItem.Enabled := selection_marked;
  PasteItem.Enabled := clipboard_not_empty;
end;
```

HyperLinkEvent

See also

Declaration : property HyperLinkEvent : THyperLinkEvent;

The hyperLinkEvent iwill be created when the User clicks on text marked as a hyperlink.

```
procedure TForm1.WPRichText1HyperLinkEvent(Sender: TObject; text,
  stamp: String; LineNumber: Longint);
var
  c : array[0..255] of Char;
begin
  StrPcopy(c,text);
  MessageBox(Handle,c,'Hyperlink clicked',0);
end;
```

The marked text will be put to `text'. stamp is reserved for future use. LineNr is the number of the line in which the selected text was found.

TWPRichText
TWPRichTextLabel

PrintFooter

See also Example

Declaration : property PrintFooter : TWPPrintEvent

The Event is created to make it possible to print on the Bottom Margin while printing with the procedure print.

You get TWPPrintEvents if you use the procedure print. It is quite easy to put graphics, page numbers or other information in the Header/Footer, because the EventHandler will get the x,y,w,h information where to print on the Canvas.

TWPRichText

```
{ Printing of the header }
procedure TForm1.WPRichText1PrintHeader(Sender: TObject;
  prCanvas: TCanvas; x, y, w, h, PageNumber: Integer; LineNumber: Longint);
begin
  prCanvas.Font.Name := 'Arial';
  prCanvas.Font.Size := 14;
  prCanvas.Font.Color:= clBlue;
  prCanvas.MoveTo(x+(wprCanvas.TextWidth('Report'))div 2,
                  y+(hprCanvas.TextHeight('A'))div 2);
  prCanvas.TextOut(x+(wprCanvas.TextWidth('Report'))div 2,
                  y+(hprCanvas.TextHeight('A'))div 2,
                  'Report');
end;

{Printing of the pagenumber  }
procedure TForm1.WPRichText1PrintFooter(Sender: TObject;
  prCanvas: TCanvas; x, y, w, h, PageNumber: Integer; LineNumber: Longint);
begin
  prCanvas.Font.Name := 'Arial';
  prCanvas.Font.Size := 12;
  prCanvas.Pen.Color := clBlack;
  prCanvas.MoveTo(x,y);
  prCanvas.LineTo(x+w,y);
  prCanvas.MoveTo(x+(wprCanvas.TextWidth(''))div 2,
                  y+(hprCanvas.TextHeight('A'))div 2);
  prCanvas.TextOut(x+(wprCanvas.TextWidth(''))div 2,
                  y+(hprCanvas.TextHeight('A'))div 2,
                  ' '+ IntToStr(PageNumber)+' ');
end;
```

TWPRuler.OnSelection

Example

Declaration : property OnSelection : TWPRulerSelectEvent;

This Event is created when the paragraph margins were changed.

The new values are in
WPRuler.Header.Layout.indentFirst
WPRuler.Header.Layout.indentLeft
WPRuler.Header.Layout.indentRight

Note: The ruler will not operate when it is not controlled by a TWPRichText component.

```
procedure TWPRichText.OnRulerSelection(Sender: TObject; var Typ:
TWpSelNr;
  var group, num: Integer; changing: Boolean);
begin
 if Assigned(FOldOnRulerSelection) then
        FOldOnRulerSelection(Sender,Typ,group,num,changing);
 if FHasFocus then
 begin
   if Typ<>wptNone then
   begin
      if Typ=wptParagraph then
      begin
        if Changing then
        begin
             if Assigned(FOnChange) then FOnChange(Self);
        end;
      end else
      if Typ=wptPage then
      begin

      end;
      Typ:=wptNone;
   end;
 end;
end;
```

OnSelection

See also Example

Declaration : OnSelection : TWPStatusBarSelectEvent

This Handler is called up when the user clicks on the StatusBar. The program tells you the ID of the string or its Index if an ID has not been set.

TWPStatusBar

```
procedure TForm1.WPStatusBar1Selection(Sender: TObject; index: Integer;
  typ: TWPStatusItemCont; name: String; x, y, w: Integer;
  var Button: TMouseButton; var Shift: TShiftState);
begin
  if typ=stUnit then
  begin
   if WPRuler1.Units = Centimeter then
      WPRuler1.Units := Inch else WPRuler1.Units:=Centimeter;
   if WPRuler1.Units = Centimeter then
     WPStatusBar1.SetString(stUnit,'Cent.')
   else WPStatusBar1.SetString(stUnit,'Inch');
  end;
end;
```

OnUpdate

See also Example

Declaration : OnUpdate : TWPStatusBarUpdateEvent

In this routine you should make the necessary initialisations.

TWPStatusBar

```
procedure TForm1.WPStatusBar1Update(Sender: TObject);
begin
   if WPRuler1.Units = Centimeter then
     WPStatusBar1.SetString(stUnit,'Cent.',TRUE)
   else WPStatusBar1.SetString(stUnit,'Inch',TRUE);
end;
```

ShiftTabPressed

See also

Declaration : property ShiftTabPressed : TNotifyEvent;

You may change the focus within an eventhandler, which is called when the user pressed Ctrl+TAB . This is useful if the property WantTabs is set to TRUE, otherwise the TAB Key can be used to jump between text objects.

TWPRichText

StartSpellCheck

See also

Declaration : property StartSpellCheck : TNotifyEvent;

StartSpellCheck will be called after the SpeckCheck button in the toolbar was clicked. You can use (Sender as TWPRichText) to determine wich Text should be checked.

SpellCheckInterface

ToolBar.OnIconSelection

See also Example

Declaration : property OnIconSelection : TWPIconSelectEvent;

The Handler for the OnIconSlection Event will be called after the user selected or deselected one of the speedbuttons.

It will get the following parameters:

```
var Typ   : TWpSelNr;
var str   : String;
var group : Integer;
var num   : Integer;
var index : Integer
```

typ coud be either wptNone or wptIconSel or wptIconDeSel.

str and index is reserved.

group defines the group of the speedbutton. num is the number within the group.
Constants for the groups and numbers are defined in WPDefs:

```
groups:
      WPI_GR_STYLE = 1;
      WPI_GR_ALIGN = 2;
      WPI_GR_EDIT  = 3;
      WPI_GR_DISK  = 4;
      WPI_GR_PRINT = 5;
      WPI_GR_DATA  = 6;
numbers:
     WPI_CO_Normal=1;   { Group: WPI_GR_STYLE }
      WPI_CO_Bold  =2;
      WPI_CO_Italic=3;
      WPI_CO_Under =4;
      WPI_CO_HyperLink = 5;
      WPI_CO_StrikeOut = 6;

      WPI_CO_Left  =1;  { Group: WPI_GR_ALIGN }
      WPI_CO_Right =2;
      WPI_CO_Justified =3;
      WPI_CO_Center=4;

      WPI_CO_Copy  =1; { Group WPI_GR_EDIT }
      WPI_CO_Cut   =2;
      WPI_CO_Paste =3;
      WPI_CO_SelAll =4;
      WPI_CO_HideSel=5;
      WPI_CO_Find   =6;
      WPI_CO_Replace=7;
      WPI_CO_SpellCheck=8;

      WPI_CO_Exit  =1; { Group: WPI_GR_DISK }
      WPI_CO_New   =2;
```

```
WPI_CO_Open   =3;
WPI_CO_Save   =4;
WPI_CO_Close =5;

WPI_CO_Print =1; { Group: WPI_PRINT }
WPI_CO_PrintSetup=2;

WPI_CO_Next    =1; { Group: WPI_DATA }
WPI_CO_Prev    =2;
WPI_CO_Add     =3;
WPI_CO_Del     =4;
WPI_CO_Edit    =5;
WPI_CO_Cancel =6;
WPI_CO_ToStart=7;
WPI_CO_ToEnd   =8;
WPI_CO_Post    =9;
```

If you handled the event and no more handling is needed please assign wptNone to typ.
To deselect a speedbutton you can call the procedure ToolBar.DeselectIcon(0, group, num);

TWPToolBar

The OnToolBarIconSelection wich is defined in TDBWPRichText.

```
procedure TDBWPRichText.OnToolBarIconSelection(Sender: TObject;
  var Typ: TWpSelNr; var str: String; var group, num, index: Integer);
begin
 if (typ=wptIconSel) then
 begin
   if (FDataLink.DataSource<>nil) and  (group=WPI_GR_DATA) then
   begin
     if (FDataLink.DataSource.DataSet<>nil)
        and (FDataLink.DataSource.State<>dsInactive)
     then
     begin
      case num of
       WPI_CO_Next   : FDataLink.DataSource.DataSet.Next; { Group: WPI_DATA
}
       WPI_CO_Prev   : FDataLink.DataSource.DataSet.Prior;
       WPI_CO_Add    : FDataLink.DataSource.DataSet.Insert;
       WPI_CO_Del    : FDataLink.DataSource.DataSet.Delete;
       WPI_CO_Edit   : if FDataLink.editing=FALSE then
                         FDataLink.DataSource.DataSet.Edit;
       WPI_CO_Cancel : FDataLink.DataSource.DataSet.Cancel;
       WPI_CO_Post   : FDataLink.DataSource.DataSet.Post;
       WPI_CO_ToStart: FDataLink.DataSource.DataSet.First;
       WPI_CO_ToEnd  : FDataLink.DataSource.DataSet.Last;
      end;
      WPToolBar.DeselectIcon(0,WPI_GR_DATA,num);
      EnableDataControlIcons;
      typ:=wptNone;
     end;
   end { Group }
   else if Focused then
   begin { handle some of the other Buttons }
       if group=WPI_GR_DISK then
         case num of
           WPI_CO_ClOSE : begin
WPToolBar.DeselectIcon(0,WPI_GR_DISK,num);
                            Typ:=wptNone; { You got the File from Database, you
can't close it ! }
                          end;
         end;
    end;
  end;
 end;
```

ToolBar.OnSelection

See also Example

Declaration : property OnSelection : TWPSelectEvent;

The handler for the OnSelection event will be called after the user selected an item of any of the listboxes in the toolbar.

The declaration of the handler has to be similar to:

```
procedure ToolBar1OnSelection(Sender: TObject;
                  var Typ: TWpSelNr;
                  str: String;
                  num: Integer);
```

typ tells wich listbox was selected:

- wptNone : ignore the call
- wptName : the font 'str' was selected
- wptSize : the fontsize 'num' was selected
- wptColor : a new color 'num' was chosen
- wptBkColor : new background color
- wptTyp : a item of the style listbox was seleted

to react on the event you should use a case expression:

```
case Typ of
            wptName :
            wptSize:
            wptTyp:
            wptColor:
            wptBKColor:
end;
```

If you handled the event and no more handling is needed please assign wptNone to typ.

TWPToolBar

This handler is defined for the TWPRichText object:

```
procedure TWPRichText.OnToolBarSelection(Sender: TObject; var Typ:
TWpSelNr;
  var str: String; var num: Integer);
var
  atr    : Tattr;
  what   : TAttrWhat;
begin
{ if Focused then  }
 begin
   if Typ<>wptNone then
   begin
      if (Changing=FALSE) or ReadOnly then begin
LastError:=ecNoChangeAllowed; exit; end;
      atr:=Attr;
      what:=[];
      case Typ of
         wptName : begin
                      atr.Font:=Memo.header.AddFontName(str);
                      what := [awFontNr];
                   end;
         wptSize: begin
                      atr.Size := num;
                      what := [awFontSize];
                  end;
         wptTyp: begin  { StyleBox }
                      { ????     }
                  end;
         wptColor: begin
                     atr.Color := (atr.Color and 240) + (num and 15);
                     what:= [awFontColor];
                   end;
         wptBKColor: begin
                       atr.Color := (atr.Color and 15) + (num * 16);
                       what:= [awFontBKColor];
                     end;
      end; { case }
      Memo.SetActAttr(atr,what);
      Typ:=wptNone;
      Changed;
      invalidate;
   end;
 end;
end;
```

Change_ParSpacing

See also

Declaration : procedure Change_ParSpacing(before,between,after:Longint);

Change_ParSpacing changes the current spacing by adding the values.
If you want to keep one value you have to pass 0, if you want to make a space larger the value has to be > 0. To shrink a space you have to pass a value <0.
Change_ParSpacing makes it possible to let the user change the spacing visually.

If you want to set special value you should better use Set_ParSpacing.

WPRichText

Clipboard Support

See also

ClipBoard functions:

procedure SelectAll;
Selects the whole text.

procedure CopyToClipboard;

The CopyToClipboard method copies the text selected to the Clipboard, replacing any text that exists there. If no text is selected, nothing is copied. The Clipboard formats CF_TEXT and Rich Text Format are supportet.

procedure CutToClipboard;

The CutToClipboard method deletes the text selected and copies it to the Clipboard, replacing any text that exists there.

procedure PasteFromClipboard;

Inserts the data of the clipboard, if there is any. The formats CF_TEXT and Rich Text Format are supportet

procedure DeleteSelection;

Deletes the selected Text.

WPRichText

Find

See also

Declaration : procedure Find(x : string;Backwards:Boolean)

procedure Find(x : string;Backwards:Boolean)

This method searches for a string. The string may contain the wildcard `*'. If you want to serach backwards, sett backwards toTRUE.
If you want to use other options bild in the Finder class you may access the property Finder. If you do so, you can replace the found text without selecting it and you may change the attributes.

procedure FindNext

finds again ...

WPRichText

Search + Replace Dialog

See also

procedure ReplaceDialog;

This procedure displayes the - non modal - replace dialog. Search and replace is allowed in both directions, as a wildcard you may use `*'.
If `replace all' take too long, the user may interrupt with ESCAPE.

procedure FindDialog;

This procedure displayes the - non modal - replace dialog. Search and replace is allowed in both directions, as a wildcard you may use `*'.

WPRichText

GetFontNr / GetFontName

See also Example

function GetFontNr(name : TFontName):Integer;

The font selection is repesented in TFont as a number. This functions tells you the number you should use.

function GetFontName(nr : Integer) :TFontName;

This Function returns the fontname if you have the number;

Attributes
WPRichText

```
var
  a      : Tattr;

{ Sets a fontname }

a.font := GetFontNr(`Arial');
WPRichText1.Attr := a;

{ Gets a Fontname }
a      := WPRichText1.Attr;
name   := GetFontName( a.Font );
```

GetTextBuf / GetSelTextBuf

See also

Declaration : function GetTextBuf(buffer : Pchar;BufSize: Longint):Longint;

The GetSelTextBuf / GetTextBuf method copies the selected/all text from WPRichText into the buffer pointed to by Buffer, up to a maximum of BufSize characters, and returns the number of characters copied.

GetSelTextBuf and the corresponding SetSelTextBuf methods use null-terminated strings that can be larger than 64K in length.
If you pass nil as pointer, only the textlenght will be calculated.

Please note:
GetSelText will insert a Carriage Return - Code at the end of each paragraph. This will cause that the lenght will exceed the lenght of SelLenght!
To calculate the required size of the buffer please use GetSelTextBuf(nil,0);

WPRichText

GetTextLen

See also

Declaration : function GetTextLen: Longint;

Thist function returns the total lenght of the text.

WPRichText

InputText

See also

Declaration : procedure InputText(s:Pchar);

With InputText you can insert Text at the cursor position. The Text will be handled like text wich was typed.

WPRichText
CPPosition

Dialog Load / Save procedures

See also

procedure SaveAs;

This procedure asks for a filename and saves then..

procedure Save;

If no filename is known (set by the load method) save will ask for a filename and save then.

procedure Load;
procedure Insert;

both procedures will asl for a filename. Insert will insert the text at cursor position, load will replace the whole Text.

Attention: No warnings are produced before overwriting the text or when the save procedure was not successfull.If you want to check if an error occur, you may after the operation, test WPRichText.LastError if it is not `ecOk'.

WPRichText

Loading and Saving

See also

function LoadFromStream(s : TStream): TWPLoadFormat; virtual;
procedure SaveToStream(s : TStream); virtual;
procedure SaveSelectionToStream(s : TStream); virtual;

These procedures load from / save to a stream.
WPRichText will use the format you define in LoadFormat/SaveFormat.
If the format is set to AutomaticFormat, WPRichText will try to find the most useful way. After calling the save procedure you may want to test LastError<>ecOk to determine if an error occur.

procedure LoadFromStreamWithReader(s : TStream;Reader:TWPTextReader);

If you want to read another Format than ANSI or RTF you may create a descendent of TWPTextReader. You may pass an instance of the new reader to LoadFromStreamWithReader.

procedure SaveToStreamWithWriter(s : TStream; Writer:TWPTextWriter);

procedure SaveSelectionToStreamWithWriter(s : TStream; Writer:TWPTextWriter);

If you want to write another Format than ANSI or RTF you may create a descendent of TWPTextWriter. You may pass an instance of the new writer to SaveToStreamWithReader.

procedure SaveToFile(const name : string);
procedure LoadFromFile(const name : string);

These functions save to file or load a file. If the filename ends with `.RTF' then SaveToFile will save in RTF Format, otherwise in ANSI Format. You Diese Funktionen schreiben in eine Datei bzw lesen von ihr.After calling the save procedure you may want to test LastError<>ecOk to determine if an error occur.

WPRichText

Print

See also

Declaration : procedure Print;

This procedure prints the text in WYSIWYG. You may write handlers to print the header and the footer of the pages: TWPPrintEvent.
During all printing WPRichText will not produce any screen output. The reason is, that the formatting of the text has been changed for the printer.

WPRichText

Print_xywh

See also

unit WPWinCtr;

function TWPCustomRtfEdit.Print_xywh(outCanvas:TCanvas;x,y,w,h:Integer;
line_nr:Longint;mode : TWPPrintMode):Longint;

This function prints on a canvas. It returns 0 if all lines could have been printed, otherwise the number of the first line which was not printed. You may use this number for another call of print_xywh.
If the option pmUsePageBreakes was set in the property Print_xywh_Mode then the function will return at the next page break.

The Print_xywh function expects:
outCanvas : the canvas to print on, for example Printer.Canvas
x,y,w,h : the Rectangle where to print on the Canvas
line_nr : the Number to start with.
mode : wpNormalPrint or
wpPrintCalc: if you only want to count the lines
wpFastPrintInit: to initialize the FastPrinting method
wpFastPrint: to print with the FastPrintig method
wpFastPrintExit: to exit Fast Printing (you should never forget to call FastPrintExit!).

Printing with FastPrint:
1. Initialize with FastPrintInit
2. Print with FastPrint as long as you need to.
3. Call FastPrintExit

Example

PutParText / InsertParText / GetParText

See also

Using this procedures you can insert or append paragraphs to the text. The new text will get the current attributes.

procedure PutParText(Index:Longint;const Buffer:Pchar;Size:Integer);

This procedure will change the paragraph # index.

procedure InsertParText(Index:Longint;const Buffer:Pchar;Size:Integer);

This procedure will insert a new paragraph before index or append it at the end.

Please note: To see the new text you have to call invalidate!

To read a paragraph you can use:

function GetParText(Index:Longint;Buffer:Pchar;Size:Integer):Integer;

This procedure will read the paragraph # index.

WPRichText
InputText
Attr

WPRtfStorage.LoadFromStream

Declaration : LoadFromStream

function LoadFromStream(s : TStream): TWPLoadFormat;

This procedures loads from a stream in RTF or ANSI format.

procedure LoadFromStreamWithReader(s : TStream;Reader:TWPTextReader);

If you want to read another Format than ANSI or RTF you may create a descendent of TWPTextReader. You may pass an instance of the new reader to LoadFromStreamWithReader.

StatusBar.SetString

See also

Declaration : function SetString(typ : TWPStatusItemCont;name : string) : Boolean;

To change the text, the statusbar is displaying you use the function SetString. If you used one of the predefined values you only have to pass the ID and the new string. The statusbar will redraw the string at once.
The function returns TRUE if the ID was found.

Valid IDs are:
'stHelp','stIndex','stOutline','stStatus','stInsert','stCaps','stNum','stXPos','stYPos','stDate','stTime','stPage','stLine','stChanged','stFont','stStyle', 'stUnit', 'stTxtFmt', 'stName', 'stFileName','stPathName'

Example:
StatusBar1.SetString(stStatus,'text loaded');

TWPStatusBar

StatusBar.SetStringIndex

See also

Declaration : function SetStringIndex(index : Integer;name : string): Boolean;

To change the text, the statusbar is displaying you use the function SetString. You have only to pass the index of the string in the property 'strings' and the new string. The statusbar will redraw the string at once.

Example:
StatusBar1.SetStringIndex(1,'field 1 changed');

TWPStatusBar

StatusBar.SetStringStyle

See also

Declaration : SetStringStyle(typ : TWPStatusItemCont;style:TWPStatusItemStyle;color:TColor)

This function defines the style and the color of the string. Style is defined as follows:

TWPStatusItemStyle = set of (sttButton, sttDown, sttDisabled, sttColor, sttBKColor);

When you set the value sttColor or sttBKColor the given color will be used for the text or the background.

If you want to access the strings with their index you can use

function SetStringStyleIndex(index : Integer;style:TWPStatusItemStyle;color:TColor) : Boolean;

TWPStatusBar

SetStringWidth

See also

Declaration : function SetStringWidth(typ : TWPStatusItemCont;width : Integer) : Boolean;

This function change the Width of an element. If the Window is too small the elements will be smaller, too, but the largest element will remain the largest one.

Valid IDs are:
'stHelp','stIndex','stOutline','stStatus','stInsert','stCaps','stNum','stXPos','stYPos','stDate','stTime','stPage','stLine','stChanged','stFont','stStyle', 'stUnit', 'stTxtFmt', 'stName', 'stFileName','stPathName'

TWPStatusBar

StatusBar.SetStringWidthIndex

See also

Declaration : function SetStringWidthIndex(index : Integer ;width : Integer) : Boolean;

This function change the Width of an element. If the Window is too small the elements will be smaller, too, but the largest element will remain the largest one. You have only to pass the index of the string in the property 'strings' and the new width.

TWPStatusBar

Set_PageSize

See also

Declaration : procedure
Set_PageSize(nmargl,nmargr,nmargt,nmarg,npagew,npageh : Longint);

With this procedure you may change the pagelayout. You can define margins and the lenght and width of the page. All numers are twips Values (1/1440 Inch):
If you don't want to change one value, just pass `KeepOldValue'.

nmargl is the left margin, nmargr the right one.
nmargt is the top margin, nmargb the bottom margin.

If you use the procedure print and want to print Headers and Footers, you have to set nmargb and nmargt to a large value: Printing of Headers and footers.
npagew defines the pagewidth
npageh defines the lenght of the paper.

WPRichText

Set_ParBorder

See also Example

Declaration : Set_ParBorder(LineType : TBorderType; Thick : Integer; Color, Space : Integer);

Set_ParBorder defines the border for the current paragraph / the selected paragraphs.

LineType can contain:
BlTop,
BLBottom,
BlLeft,
BlRight,
BLBox,
BLDouble,
BLDot,
BlTabLines

Thick is in twips the width of the border line. The smallest unit is 1pt = 20 twips.

Color is the index of the color (0..15)

Space is the space between text and border in mm.

WPRichText
Set_ParBorder

{ part of the source of TWPParagraphBorderDlg }

```
procedure TWPParagraphBord.BitBtn2Click(Sender: TObject);
var
  typ : TborderType;
  col,wid,spc : Integer;
begin
  if fsEditBox<>nil then
  begin
    typ := [];
    if box.Checked then include(typ,blBox);
    if topbox.Checked then include(typ,blTop);
    if leftbox.Checked then include(typ,blleft);
    if bottombox.Checked then include(typ,blbottom);
    if rightbox.Checked then include(typ,blright);
    if double.Checked then include(typ,blDouble);
    if dotted.Checked then include(typ,blDot);
    if typ<>[] then
    begin
      if ColorText.Text<>'' then
         col := StrToInt(ColorText.Text)
         else col:=0;
      if SpaceText.Text<>'' then
         spc := StrToInt(SpaceText.Text)
         else spc:=0;
      if WidthText.Text<>'' then
         wid := StrToInt(WidthText.Text)*20
         else wid :=1;
      fsEditbox.Set_ParBorder(typ, wid, col,spc);
    end
    else fsEditbox.Set_ParBorder([], 0, 0, 0);
  end;
  ModalResult := IDOK;
end;
```

Set_ParBorderFinish

See also

Declaration : procedure TWPRichText.Set_ParBorderFinish;

Set_parBorderFinish markes the current paragraph to be the last paragraph with a border. Set_ParBorderFinish will not delete the border. To delet a border you can call Set_ParBorder([],0,0,0).

TWPRichText
TAttr
Set_ParBorder

Set_ParMargin

See also

Declaration : procedure Set_ParMargin(first,left,right:Longint);

With this procedures you can set the indentation of the current paragraph. The values are expected to be twips and to be in the range form 0 to 32000 if you are using Delphi 1 (16 Bit). With Delphi 2 ther is no limitation.

WPRichText

Set_ParSpacing

See also

Declaration : procedure Set_ParSpacing(before,between,after:Longint);

This procedure will set the new spacing values for the current paragraph or the markes paragraphs.

Before is the space in twips before a paragraph.

Between is the lineheight. If it is 0 the tallest character will be responsible for the lineheight. (automatic lineheight)

After is the space after a paragraph.

If you don't want to change one value you may pass KeepOldValue.

The values are expected to be twips and to be in the range form 0 to 32000 if you are using Delphi 1 (16 Bit). With Delphi 2 ther is no limitation.

WPRichText

SetPosition / GetPosition

See also

procedure SetPosition(pos,lin,par,pos_in_par:Longint);

This procedure will change the cursorposition.
You may use
pos, as an absolute textposition util the beginning of the text. - or -
lin, as a Linenumber -or -
par as a paragraph numer with pos_in_par as the position inside of the paragraph.

procedure GetPosition(var pos,lin,par,pos_in_par:Longint);

This procedure will tell you the cursorposition.

WPRichText

SetSelTextBuf

See also

Declaration : procedure SetSelTextBuf(s : Pchar);

The SetSelTextBuf method sets the selected text in the WPRichText control to the text in the null-terminated string pointed to by Buffer. SetSelTextBuf and the corresponding GetSelTextBuf methods use null-terminated strings that may be larger than 64KB. SetSelTextBuf and SetSelText will read the buffer as RTF if it starts with {rtf !

WPRichText
SetTextBuf

SetTextBuf

See also

Declaration : procedure SetTextBuf(s : Pchar);

The SetTextBuf method replaces all text in the WPRichText control to the text in the null-terminated string pointed to by Buffer. SetSelTextBuf and the corresponding GetSelTextBuf methods use null-terminated strings that may be larger than 64KB. SetSelTextBuf and SetSelText will read the buffer as RTF if it starts with {rtf !

WPRichText
SetSelTextBuf

SpellCheck Interface

See also Example

procedure Spell_FromStart;

This procedure will move the spellcheck position to the start of the text.

procedure Spell_FromCursorPos;

If you call this procedure, the next time when using GetTextWord, the word after the cursor position will be returend.

function Spell_GetNextWord : string;

The next word will be returned. If there is no more text, `' will be returned.

procedure Spell_ReplaceWord(s : string);

The last returned text will be replaced invisibly. If you have a spelling dialogbox you should not use this function. It is better to use SelectWord and then change the word by using SelText := NewWord.

procedure Spell_SelectWord;

This procedure will mark the last word and set the cursor positon after this word.

Start SpellCheck Event

```
{ This source can be used as a template to build a
  non modal spell check dialog }

unit Wpspdlg1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, WPWinCtr;

type
  TWPSpDialog = class(TForm)
    WordList: TListBox;
    Word: TEdit;
    ReplaceAs: TEdit;
    Ignore: TBitBtn;
    Replace: TBitBtn;
    Cancel: TBitBtn;
    procedure IgnoreClick(Sender: TObject);
    procedure ReplaceClick(Sender: TObject);
    procedure CancelClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    fsFirstCall : Boolean;
    fsEditBox   : TWPCustomRtfEdit;
    procedure SetEditBox(x : TWPCustomRtfEdit);
  public
    property EditBox : TWPCustomRtfEdit read fsEditBox write SetEditBox;
  end;

TWPSpellCheckDlg = class(TComponent)
private
  dia       : TWPSpDialog;
  fsEditBox : TWPCustomRtfEdit;
public
  destructor  Destroy;
  procedure Execute;
published
  property EditBox : TWPCustomRtfEdit read fsEditBox write fsEditBox;
end;

var
  WPSpDialog: TWPSpDialog;

implementation
```

```
{$R *.DFM}

destructor TWPSpellCheckDlg.Destroy;
begin
  if dia<>nil then
  begin
        if dia.visible then dia.Close;
        dia.Free;
  end;
end;

procedure TWPSpellCheckDlg.Execute;  { Will not free the Dialogbox because
it is nnon modal ! }
begin
 if assigned(fsEditBox) then
 begin
   if dia=nil then dia := TWPSpDialog.Create(Self);
   dia.EditBox := fsEditBox;
   fsEditBox.Spell_FromCursorPos;
   dia.Show;
 end;
end;

procedure TWPSpDialog.SetEditBox(x : TWPCustomRtfEdit);
begin
  fsEditBox := x;
  if x<>nil then x.Spell_FromCursorPos;
end;

procedure TWPSpDialog.IgnoreClick(Sender: TObject);
var
 s : String;
begin
 while TRUE do
 begin
  EditBox.Spell_FromCursorPos;
  s := EditBox.Spell_GetNextWord;
  if s='' then break;
  { test Dictionary .... ##################################
    if not InDictionary(s) then
    begin
       EditBox.Spell_SelectWord;
       WordList.Strings.Assign( GuessList );
       break;
    end
```

```
    else continue;

#########################################################
######## }
   { without a dictonary any word is unknown: }
      EditBox.Spell_SelectWord;
      break;
 end;
 Word.Text      := s;
 ReplaceAs.Text := s;
 if (s='') and (fsFirstCall=FALSE) then close;
end;

{ Replace will replace the selected Text.
  If you use the procedure
  EditBox.Spell_ReplaceWord(s : string) the replacment will be done
  invisibly.
  If you have a non modal spell dialog, the use of Spell_ReplaceWord might
  cause errors if the user has changed the text meanwhile.
}

procedure TWPSpDialog.ReplaceClick(Sender: TObject);
begin
  if CompareStr(EditBox.SelText,Word.Text)=0 then  { Replace selceted Text }
     EditBox.SelText := ReplaceAs.Text;
  IgnoreClick(Sender);
end;

procedure TWPSpDialog.CancelClick(Sender: TObject);
begin
  EditBox.Spell_FromCursorPos;
  Close;
end;

procedure TWPSpDialog.FormShow(Sender: TObject);
begin
  fsFirstCall := TRUE;
  IgnoreClick(Sender);
  fsFirstCall := FALSE;
end;

end.
```

ToolBar.AddControl

See also Example

Declaration : procedure AddControl(comp : TControl);

You may put any component to the others in the toolbar using AddControl. If you want to remove the component you have to call RemoveControl.

procedure AddControl(comp : TControl);
procedure RemoveControl(comp : TControl);

TWPToolBar

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  WPToolBar1.AddControl(Preview);
end;
```

ToolBar.GetIcon - StyleBox - GetFontBox ...

See also Example

Declaration : function GetIcon(index,group,num:Integer):TSpeedButton;

You may want to change some properties of the components wich are used by the toolbar. To provide this possibility, the toolbar can return the object you are asking for. Please test if the return value is nil, before using it.

GetIcon(index,group,num:Integer):TSpeedButton;

You have only to pass the group and number of the button you want to access.
Please note that you may change nearly any property, but not the Tag.

GetStyleBox : TComboBox;
GetStyleItems: TStrings;
GetFontBox : TComboBox;
GetFontItems : TStrings;
GetSizeBox : TComboBox;
GetSizeItems : TStrings;

TWPToolBar

```
{ define Hints for the SpeedButtons in the Toolbar }

procedure TForm1.FormCreate(Sender: TObject);
type
 TOneHint = record
   group, number : Integer;
   Hint          : String;
  end;

var
 i : Integer;
 button : TSpeedButton;

const Hints : array[1..6] of TOneHint =
((group : WPI_GR_STYLE; number : WPI_CO_Normal; hint : 'reset styles'),
 (group : WPI_GR_STYLE; number : WPI_CO_Bold; hint : 'bold'),
 (group : WPI_GR_STYLE; number : WPI_CO_Italic; hint : 'italic'),
 (group : WPI_GR_STYLE; number : WPI_CO_Under; hint : 'underlined'),
 (group : WPI_GR_STYLE; number : WPI_CO_Hyperlink; hint : 'hyperlink style'),
 (group : WPI_GR_STYLE; number : WPI_CO_StrikeOut; hint : 'strike out'));

begin
  for i:=1 to 6  do with Hints[i] do
  begin button := WPToolBar1.GetIcon(0,group,number);
        if button<>nil then
        begin button.Hint     := Hint;
              button.ShowHint := TRUE;
        end;
  end;
end;
```

ToolBar.SelectIcon,DeselectIcon,EnableIcon

See also Example

Declaration : function SelectIcon(index,group,num:Integer):Boolean;

With this function you can set the down property of any speedbutton to true or to false. With EnableIcon you can disable/enable any speedbutton.

SelectIcon(index,group,num:Integer):Boolean;
DeselectIcon(index,group,num:Integer):Boolean;
EnableIcon(index,group,num:Integer;enabled:Boolean):Boolean;

TWPToolBar

```
procedure TForm1.UpdateTableState;
begin
  with Table1 do
  begin
    WPToolBar1.EnableIcon(0,WPI_GR_DATA,WPI_CO_Next,not EOF);
    WPToolBar1.EnableIcon(0,WPI_GR_DATA,WPI_CO_ToEnd,not EOF);
    WPToolBar1.EnableIcon(0,WPI_GR_DATA,WPI_CO_Prev,not BOF);
    WPToolBar1.EnableIcon(0,WPI_GR_DATA,WPI_CO_ToStart,not BOF);
    WPToolBar1.EnableIcon(0,WPI_GR_DATA,WPI_CO_Edit,not Readonly);
  end;
end;
```

WriteStatus

See also

Declaration : procedure WriteStatus(s:Pchar);

Writes a message to the statuslabel or to the string `stStatus' in the StatusBar.

WPRichText

Attr

See also Example

Declaration : property Attr : TAttr;

Attr is the current writing mode. You may change the writing mode by assigning a TAttr structure. If currently a selection was made the change will be applied to the selection.

TAttr
DataStructures

```
TForm1.Button1Click(Sender : TObject)
var
  a : Tattr;
  i : Integer;
begin
  a := WPRichText1.Attr;
  a.Font := WPRichText1.GetFontNr('arial');
  a.Size := 4;
  a.Style := [afsItalic];
  i := 0;
  while i<Memo1.Lines.Count do
  begin
     WPRichText1.Attr := a;
     WPRichText1.Lines.Add(Memo1.Lines.[strings]);
     a.Size := a.Size + 2;
     inc(i);
  end;
end;
```

BackgroundColor

See also

Declaration : property BackgroundColor : TColor;

With BackgroundColor you could - best at runtime - set the color wich will be used for new text. If a selection was made, the backgrund color will be set for all selected characters.
If you use the Lines property to assign text, the given BackGroundColor will be used to.

TWPRichText

CPLine

See also

Declaration : property CPLine : string

CPLine can be used to get the text of the current line as a string. A string may be assigned, too. Because word wrap will be applied at once, in this example string a will (most times) not be like string b!

```
var
  a,b : String;
begin
  a := 'Hello';
  WPRichText1.CPLine := a;
  b := WPRichText1.CPLine;

  if CompareText(a,b)=0 then
  begin
     { just luck   }
  end
end;
```

TWPRichText

CPLineNr

See also

Declaration : property CPLineNr : Longint;

CPPosition can be used to determine the line number of the current line or to set the actual line by number.

Note:
To acces the data of the current line you may use the working parameters Memo.active_paragraph ans Memo.active_line. See datastructures, too.

TWPRichText

CPPosition

See also

Declaration : property CPPosition : Longint

CPPosition can be used to determine the Cursor position (in characters from beginning of the text) or to change the cursor position.

TWPRichText

CaretDisabled

See also

Declaration : property CaretDisabled : Boolean;

if CaretDisabled = TRUE then the TWPRichText or TDBWPRichText object will not show a caret, although the actual writing position still logically exist.

TWPRichText

EditBox

See also Example

Declaration : property EditBox : TWPCustomRtfEdit;

EditBox defines the WPRichText or DBWPRichText component wich contains the text wich has to be changed by the components

TWPRichText
TWPPagePropDlg
TWPParagraphBorderDlg
TWPParagraphPropDlg

```
try
WPPagePropDlg := TWPPagePropDlg.Create(Self);
WPPagePropDlg.EditBox := WPRichText1;
WPPagePropDlg.Execute;
finally
WPPagePropDlg.Free;
end;
```

ExternHorScrollBar

See also

Declaration : property ExternHorScrollBar : TScrollBar;

ExternHorScrollBar can be used if you want to have a scrollbar wich is not near the RichText window or has a different Width.

TWPRichText

ExternVerScrollBar

See also

Declaration : property ExternVerScrollBar : TScrollBar;

ExternVerScrollBar can be used if you want to have a scrollbar wich is not near the RichText window or has a different Height.

TWPRichText

WPRichText.Finder

See also Example

Declaration : property Finder : TWPTextFinder ;

Finder allowes to get and to set the object wich handles searching and replacing. It has of type TWPTextFinder and you may create a new one and assign it to Finder. This makes it possible to search in any way.
If you access Finder directly, you have more options than by only using the Find procedure.

Example:

```
procedure TForm1.MarkDatafields;
var
  newattrr: TAttr;
begin
  newattr := WPRichText1.Attr;
  newattr.Style := afsItalic;

  WPrichText1.Finder.WildCard := '*';
  WPrichText1.Finder.ToStart;
  while WPRichText1.Finder.Next('[@*@]') do
  begin
      WPRichText1.Finder.foundAttr := newattr;
  end;
end;
```

TWPRichText

```
{ Definition of the Finder Class }

WPTMatchAttr = set of (wmtFont,wmtSize,wmtBold,wmtUnderlined,wmtItalic,
 wmtHyperlink,wmtColor,wmtBkColor);
WPTMatchChar = (wpmtNormal,wpmtMatchCase);

TWPTextFinder = class
protected
  prior_lin  : PTLine;      { Searching variables. Will be restored using  }
  prior_par  : PTParagraph; { act_pos MoveTo(pos)  }
  prior_pa   : PTAttr;
  prior_pc   : Pchar;
  prior_cp   : Integer;  { Position in Line }
  prior_pos  : Longint;  { changed during search }
  {-------------------------------------------------------}
  actpos  : Longint;  { Last Position                     }
  actlen  : Integer;  { If found, then Lenght of found text }
  {-------------------------------------------------------}

  DidFind     : Boolean; { Something found          }
  endOfText   : Boolean; { End of Text and not found }
  backwards   : Boolean; { Direction to serach       }
  comp_text   : array[0..256] of Char;

  last_found_string : String;
  last_found_attr   : TAttr;
private
  function   GetFoundPos   :  Longint;  { returns the position from start
}
  function   GetFoundLen   :  Longint;  { Length of the Text (this time just on
one line!  }
  function   GetFoundText:  String;   { will try to put found text to a string
}
  procedure  SetFoundText(s : String);{ will replace the found text and keep it
as found }
  function   GetFoundAttr  :Tattr;      { will return the FIRST attr of the found
text     }
  procedure  SetFoundAttr(s :Tattr);    { will set ALL attr of the found text
}
  procedure  SetActPos(x : Longint);
protected
  function   MoveTo(pos : Longint):Longint; virtual;
  procedure  MoveNext(s : string); virtual;
  procedure  MoveBack(s : string); virtual;
  function   CompareChar(ca,cb:Char;apa,bpa:PTattr):Boolean; virtual;
{ FALSE if not equal }
```

```
  function   EndAtWordChar(c : Char):Boolean; virtual;
public
  { Searches in the text forewards: }
  function    Next(s : String): Boolean; virtual; { TRUE if found    }
  { Seraches in the text backwards: }
  function    Prev(s : string): Boolean; virtual; { search backwards }
  { Moves Serach position to End/Start of the Text }
  procedure   ToEnd;
  procedure   ToStart;
  { moves the start position od the found text }
  procedure   MoveFoundPos(diff : integer);
public      { These properties are set befor the call of WriteHeader /
WriteText }
  FHeader       : TTextHeader; { for page and font definition }
  fsWordProc    : TWPRTFText; { needed for replace }
  first_par     : PTParagraph; { The first paragraph }
  match_attr    : WPTMatchAttr; { may be () }
  match_char    : WPTMatchChar;
  wildcard      : Char; { #0 or wildcard }
  endchar       : Char; { #0 to scan to end, or Whitespace or whatever (only
forward scanning!) }
  EndAtWord     : Boolean; { if true than stop at '.,?!")' ... }
  cmp_attr   : Tattr;  { attributes to match WPTMatchAttr }
  property     Memo : TWPRTFText read fsWordProc write fsWordProc;
  property     position : Longint write SetActPos;
  property     foundPosition : Longint read GetFoundPos;
  property     foundLength : Longint read GetFoundLen;
  property     foundText: String  read GetFoundText write SetFoundText;
  property     foundAttr: Tattr read GetFoundAttr write SetFoundAttr;
end;
```

FitToWindowHorz

See also

Declaration : property FitToWindowHorz : Boolean;

If this property is TRUE, the text will be formatted to the window width. The page layout will be ignored for screen output, but not for printer output.

TWPRichText
TWPRichTextLabel

HyperLinkCursor

See also

Declaration : property HyperLinkCursor : TCursor;

Choose the kind of the cusror wich will be used when the cursor (mouse!) is near a text wich is marked as a hyperlink. If the value is crDefault it will save probably some CPU usage.

TWPRichText
TWPRichTextLabel

HyperLinkEvent

See also Example

Declaration : property HyperLinkEvent : THyperLinkEvent;

Select your Eventhandler for the HyperLink Event. You can for example jump to another position in th text or in the database.

WPRichText will pass the following parameters:
text : String = the marked text
stamp: String (reserved for future use)
LineNumber: Longint

TWPRichText
TWPRichTextLabel

```
procedure TForm1.WPRichText1HyperLinkEvent(Sender: TObject; text,
  stamp: String; LineNumber: Longint);
var
  c : array[0..255] of Char;
begin
  StrPcopy(c,text);
  MessageBox(Handle,c,'Hyperlink clicked',0);
end;
```

IsLastLine

See also

Declaration : property IsLastLine : Boolean;

IsLastLine is TRUE when the cursor is in the last text line.

TWPRichText

TWPRichTextLabel.Transparent

See also

Declaration : property Transparent : Boolean;

The Transparent property determines if a rich text label is transparent. You could place a transparent label on top of a bitmap, and the control won't hide part of the bitmap. The given background colour will be ignored.

TWPRichTextLabel

Language

See also

Declaration : property Language : TWPToolLanguage;

You can choose between

twpEnglish oder twpGerman

The value is saved in a global variabled defined in unit WPDefs:

var
GlobalLanguage : TWPToolLanguage;

TWPRichText

LastError

See also

Declaration : property LastError : ecErrCode;

unit WPDefs;

If LastError<>ecOK thenan error occured during the last Fileoperation.

TWPRichText

Lines

See also Example

Declaration : property Lines : TStrings;

At Runtime only!

This property corresponds to the Lines property of a TMemo or a TListBox component. Since only non formatted text is stored text and is limited to the length 255 this property can be only used limited. Also Lines.Strings [x] will not address the line x, but the paragraph x! The paragraphs are often longer than 255 characters, too! The Lines property is suitable to transfer the data of a TMemo or TListBox component but when transfering text from a TWPRichText component to a TMemo component text may get lost.

When inserting Text you may change the writing mode to set the attributes of the new text.

TWPRichText

```
TForm1.Button1Click(Sender : TObject)
var
  a : Tattr;
  i : Integer;
begin
  a := WPRichText1.Attr;
  a.Font := WPRichText1.GetFontNr('arial');
  a.Size := 4;
  a.Style := [afsItalic];
  i := 0;
  while i<Memo1.Lines.Count do
  begin
     WPRichText1.Attr := a;
     WPRichText1.Lines.Add(Memo1.Lines.[strings]);
     a.Size := a.Size + 2;
     inc(i);
  end;
end;
```

LoadFormat

See also

Declaration : property LoadFormat : TWPLoadFormat

LoadFormat tells the loading procedure in wich format the text is in the stream or th file.

TWPLoadFormat =
 AutomaticFormat
 LastFormat
 FormatRTF
 FormatANSI

AutomaticFormat is best, because the procedure will check if the data starts with '{\rtf' and then load in RTF format, otherwise in ANSI format.

TWPRichText
Load + Save

MemoryFormat

See also

Declaration : property MemoryFormat : TWPMemoryFormat

You may choose between fmRichText or fmPlainText. If you switch from fmRichText to fmPlainText you will loose all formattings except of indenting, spacing and borders. If you have fmPlainText and is very large you may not have enough RAM to switch to fmRichText. If the process takes too long, the user may cancel with the ESCAPE key.

Please note: The fmRichText Format uses about additional 10 Bytes for each character.

TWPRichText

Modified

See also Example

Declaration : property Modified : Boolean;

Runtime Only!

The property Modified may be used to determine if the Text in the RichText Component was changed.
The property modified may be changed, too.

TWPRichText

```
{ save the text }
if WPRichText.Modified then
begin
    WPRichText.Save;
    if WPRichText.LastError=ecOK then
            WPRichText1.Modified := FALSE;
end;
```

NoBlockMarking

See also

Declaration : property NoBlockMarking : Boolean

If NoBlockMarking is TRUE no Slections are allowed.

TWPRichText

OneClickHyperlink

See also

Declaration : property OneClickHyperlink : Boolean

If OneClickHyperlink is TRUE the user can start a hyperlink with one click instead of a doubleclick.

TWPRichText

Print_xywh_Mode

See also Example

Declaration : property Print_xywh_Mode : TWPPrint_XYWH_Mode;

Using Print_xywh_Mode you can change the way how prining on an canvas object is done.

```
TWPPrint_XYWH_Mode = Set of   (     pmUsePaper,
      pmWhiteTransparent,
      AllTransparent,
      pmUseBKColor,
      pmIgnoreZooming,
      pmUsePageBreaks);
```

pmUsePaper : deletes empty spaces
pmWhiteTransparent : the default backgrounf color will be transparent
pmAllTransparent : all text is transparent
pmUseBKColor : the default backgrund color will be printed
pmIgnoreZooming : zooming is not used.
pmUsePageBreakes : normaly print_xywh prints sp many lines as fit into the rectangle. With this option it stops at the next page break.

TWPRichText
Print_XYWH

```
{ Example for zoomable Print Preview }

unit PrinPr;
{ This Unit shows how to use buttons and strings in the statusbar
  and how to use the function print_xywh(Canvas, ...

  I Use the wpFastPrint Method in this unit:
    1. Call   print_xywh(Canvas, ... , wpFastPrintInit)
    2. Then call  print_xywh(Canvas, ... , wpFastPrint)   so often as you like
    3. Call   print_xywh(Canvas, ... , wpFastPrintExit)
You shoul NEVER forget to call wpFastPrintExit !
The FastPrint Method is much faster wich large Text with lots of pages to print
because initialisation is only done one time.

If you have only short Text ore more time
then you may call print_xywh(Canvas, ... , wpNormalPrint) and you dont have
to worry.

If you want to check if there is enoug space to print all lines you can call
print_xywh(Canvas, ... , wpPrintCalc). You may do this after wpFastPrintInit,
too.
}

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, WPRich, WPWinCtr, WPStatus,
WPDefs;

type
  TPrintPreview = class(TForm)
    WPStatusBar1: TWPStatusBar;
    Panel1: TPanel;
    PaintBox1: TPaintBox;
    procedure FormCreate(Sender: TObject);
    procedure WPStatusBar1Update(Sender: TObject);
    procedure PaintBox1Paint(Sender: TObject);
    procedure WPStatusBar1Selection(Sender: TObject; index: Integer;
      typ: TWPStatusItemCont; name: String; x, y, w: Integer;
      var Button: TMouseButton; var Shift: TShiftState);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Resize(Sender: TObject);
  private
    { Private-Deklarationen }
    initialized : Boolean; { wpFastPrintInit called ? }
```

```
    Zoom    : Integer;
  public
    { Public-Deklarationen }
    last_line,first_line : Longint;
    WordProcessor : TWPRichText;
  end;

var
  PrintPreview: TPrintPreview;

implementation

{$R *.DFM}

procedure TPrintPreview.FormCreate(Sender: TObject);
begin
  WPStatusBar1.SetStringWidthIndex(0,75);
  WPStatusBar1.SetStringWidthIndex(1,75);
  WPStatusBar1.SetStringWidthIndex(1,75);
end;

procedure TPrintPreview.WPStatusBar1Update(Sender: TObject);
begin
  WPStatusBar1.SetStringIndex(0,IntToStr(first_line));
  WPStatusBar1.SetStringIndex(0,'');
  WPStatusBar1.SetStringIndex(0,'');
end;

procedure TPrintPreview.PaintBox1Paint(Sender: TObject);
begin
   if not initialized then                              { I Use the FastPrintMethod ! }
   begin WordProcessor.Print_xywh(PaintBox1.Canvas,0,0,
            PaintBox1.Width,PaintBox1.Height, first_line,wpFastPrintInit);
         initialized := TRUE;
   end;

   { Set zooming }
   WordProcessor.Zooming := Zoom;

   { Options: pmUsePaper,pmWhiteTransparent,
        pmAllTransparent,pmUseBKColor,pmIgnoreZooming,
        pmUsePageBreaks
     all in unit WPWinCtr. }

   WordProcessor.Print_XYWH_Mode := [pmUsePageBreaks];
   last_line := WordProcessor.Print_xywh(PaintBox1.Canvas,0,0,
```

```
        PaintBox1.Width,PaintBox1.Height, first_line,wpFastPrint);

   with PaintBox1.Canvas do
   begin
     Pen.Width := 1;
     Pen.Color := clBlack;
     Pen.Style := psDot;
     MoveTo(0,0);
     LineTo(PaintBox1.Width-1,0);
     LineTo(PaintBox1.Width-1,PaintBox1.Height-1);
     LineTo(0,PaintBox1.Height-1);
     LineTo(0,0);
   end;

   { The last FALSE means that output shoud be done, not only calculation }
end;

procedure TPrintPreview.WPStatusBar1Selection(Sender: TObject;
  index: Integer; typ: TWPStatusItemCont; name: String; x, y, w: Integer;
  var Button: TMouseButton; var Shift: TShiftState);
begin
   if index=3 then Close { selected END }
   else if index=2 then   { selected NEXT }
   begin
        first_line := last_line;
        PaintBox1.invalidate;
        WPStatusBar1.SetStringIndex(0,IntToStr(first_line));
   end;
end;

procedure TPrintPreview.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
   if initialized then                              { Never forget to call Exit !  }
   begin WordProcessor.Zooming := 100;
        WordProcessor.Print_xywh(PaintBox1.Canvas,0,0,
            PaintBox1.Width,PaintBox1.Height, first_line,wpFastPrintExit);
        initialized := TRUE;
   end;
end;

procedure TPrintPreview.Resize(Sender: TObject);
var
  xm, ym : Real;
  papxsiz, papysiz,lmarg,tmarg : Longint;
  x,y,w,h : Integer;
```

```
begin
  papxsiz := (WordProcessor.Memo.Header.Layout.paperw *
       Screen.PixelsPerInch) div 1440;
  papysiz := (WordProcessor.Memo.Header.Layout.paperh *
       Screen.PixelsPerInch) div 1440;
  xm := Width / papxsiz;
  ym := (Height-WPStatusBar1.Height-50) /  papysiz;
  if xm < ym then ym := xm;

  w := round(papxsiz * ym);
  H := round(papysiz * ym);

  x := (Width - Panel1.Width) div 2;
  y  := (Height -(WPStatusBar1.Height + h + 10)) div 2;

  Panel1.SetBounds(x,y,w,h);

  lmarg := (WordProcessor.Memo.Header.Layout.margl *
       Screen.PixelsPerInch) div 1440;
  tmarg := (WordProcessor.Memo.Header.Layout.margt *
       Screen.PixelsPerInch) div 1440;

  y := round(tmarg*ym);
  x := round(lmarg*ym);
  lmarg := (WordProcessor.Memo.Header.Layout.margr *
       Screen.PixelsPerInch) div 1440;
  tmarg := (WordProcessor.Memo.Header.Layout.margb *
       Screen.PixelsPerInch) div 1440;

  w :=  w - x - round(lmarg*ym);   { w= Panel1.Width }
  h :=  h - y - round(tmarg*ym);   { h= Panel1.Height }

  PaintBox1.SetBounds(x,y,w,h);

  if not Panel1.Visible then Panel1.Visible:= TRUE;
  Zoom := round(100 * ym);

  WPStatusBar1.SetString(stStatus,IntToStr(Zoom)+'%');
end;

end.
```

WPRuler.ShowAllTabPos

See also

Declaration : property ShowAllTabPos : Boolean

When set to TRUE the tab positions wich are defined but not used in the current paragraph are shown in gray color.

TWPRuler

WPRuler.Units

See also

Declaration : property Units : PTWPRulerUnit;

You can choose the unit for the ruler:

Centimeter or Inch

You won't need to use any other properties or events because the TWPRichText component will control the ruler.

TWPRuler

WhileSelection

See also

Declaration : property WhileSelection : TWPRulerSelectEvent;

This Event is created while the user changes the paragraph margins.

The new values are in
WPRuler.Header.Layout.indentFirst
WPRuler.Header.Layout.indentLeft
WPRuler.Header.Layout.indentRight

Note: The ruler wil not operate when if it is not controlled by a TWPRichText component.

TWPRuler

WPRuler.XOffset

See also

Declaration : property XOffset : Integer;

XOffset scrolles the ruler.

TWPRuler

Resizing

See also

Declaration : property Resizing : Integer;

With Resizing you may tell the RichText component how much the text should be enlarged or shrinked. The resolution will not be changed (unlike Zooming).
Resizing enlarges the textfonts (instead changing the MapMode) and will reformat the complete text.
You can use Resizing when the current text does not fit on one page.

WPRichText

RtfText

See also

Declaration : property RtfText : TWPRTFTextIO;

You can edit the Text in the property RtfText at designtime. You can define the pagelayout and modify the color table. The changes will be saved if the lenght of the text is a least 1 char. You can delete all text if you overwrite 'RTF Text...' in the Object Inspector with 'empty!'.
You can assign RTF Text from another RichText Control:

```
WPRichText1.RtfText.Assign(WPRichText2.RtfText);
```

useful is to store the Text in the invisible TWPRtfStorage object and assign it when needen:

```
WPRichTextLabel1.ButtonClick(Sender:TObject);
begin
    WPRichTextLabel1.Visible := FALSE;
    WPRichTextLabel1.RtfText.Assign( WPRtfStorage1.RtfText);
    WPRichTextLabel1.Visible := TRUE;
end;
```

Attention: Please use the option of the property editor to save your text.

TWPRtfStorage
TWPRichText
TWPRichTextLabel

The Gauge in the StatusBar

See also Example

Didn't you think about a gauge which makes it possible to show a gauge in a subroutine, without knowing if the subroutine is the only routine which was called, or if it takes only 1/10 of the time, the whole operation will take?

Example:
Supposed you want to copy a file and show a gauge. Thats easy. Calculate the length of the file and increment the gauge with the actual position.

```
procedure filecopy(name:string)
var     length, pos    : Longint;
 begin
        length := lengthOfFile(name)
        GaugeIn(length)
        for pos:=0 to length do
        begin { copy }
             GaugeInc(pos);
        end;
        GaugeOut;
end;
```

Now you want to copy lots of files.

```
procedure Copy(names : array[0..10] of string)
 var i : Integer;
         AllCount,ActCount : Longint;
 begin
         AllCount :=0;
         for i:=0 to 10 do AllCount := AllCount +   lengthOfFile(name[i]);
         GaugeInc(count);
         for i:=0 to 10 do
         begin ActCount := lengthOfFile(name[i]);
                     GaugeRestrictSub(100 * (ActCount / AllCount) );
                     FileCopy(name[i]); { this is the procedure above! }
         end;
         GaugeOut;
end;
```

The use is that you now you can use a gauge in a procedure and use this procedure as a subroutine or as as a sub-subroutine.

procedure GaugeReset;
Resets all values.

function   GaugeIn(end_value   : Longint) : Longint;
Initialise new parameters on the gauge stack.End_value is the maximum expected value.

procedure GaugeRestrictSub(val : Integer);
Any called subroutine may only use val % of the gauge.

procedure GaugeInc(act_value : Longint);

Increment the gauge to position act_value.

procedure GaugeOut;
Destroys the paremeters which were last initialised with GaugeIn.

TWPStatusBar

```
procedure DoSomething;
var
  pos : Integer;
  procedure SubRoutine1(actpos : Integer);
  var
    a,b : Longint;
  begin
    b := SizeOf(Block[actpos]);
    GaugeIn(b);
    for a:=0 to b do
      begin
        ..... do something with Block[actpos]
        GaugeInc(a);
      end;
    GaugeOut;
  end;
begin
  GaugeIn(10);
  GaugeRestrictSub(10);  you know, that one call of subroutine1
       takes 10% of the time for the DoSomething will take.
  for pos:=1 to 10 do
      subroutine1(pos);
  GaugeOut;
end;
```

GaugeValue

See also

Declaration : GaugeValue : Integer

This Value has to be between 0 and 100 %.

TWPStatusBar
Gauge

GaugeWidth

See also

Declaration : GaugeWidth : Integer

This is the width of the gauge. If the Window becomes too small the gauge will be made smaller, but the relation of the widths will remain.

TWPStatusBar
Gauge

ShowGauge

See also

Declaration : ShowGauge : Boolean

TRUE, to show the gauge.

TWPStatusBar
Gauge

ShowSizer

See also

Declaration : ShowSizer : Boolean;

Do you like Win 95?

TWPStatusBar

Strings

See also

Declaration : Strings : TStringList

At designtime and at runtime you may create strings. To make the access of the strings easier you have to use the predefined (st...) Values. You may use any word, but than you hav to access the text later using the index.
If you want to change any status information you should not use the Strings property. You should change the text using `SetString .... `!

This are the valid IDs:

'stGauge','stHelp','stIndex','stOutline','stStatus','stInsert','stCaps','stNum','stXPos','stYPos','st
Date','stTime','stPage','stLine','stChanged','stFont','stStyle',
'stUnit','stTxtFmt','stName','stFileName','stPathName','stString'

TWPStatusBar

StringsAlign

See also

Declaration : StringsAlign : TWPStatusItemAlign

How to align the strings: alTop, alCenter or alBottom.

TWPStatusBar

StusBar.UseGaugeForWP

See also

Declaration : property UseGaugeForWP : Boolean ;

If UseGaugeForWP is set to TRUE the statusbar will get all gauge events of all richtext controls in the project.
You may of course use the gauge in the statusbar, too, but the result is not defined.

TWPStatusBar
Gauge

SaveFormat

See also

Declaration : property SaveFormat : TWPLoadFormat

Save Format defines the way the rich text control will save the text. You can choose ANSI or RTF format.

TWPLoadFormat =
 AutomaticFormat
 LastFormat
 FormatRTF
 FormatANSI

AutomaticFormat:
When saving to file the extension of the file is used to determine wich format should be used.
In other cases the last loading format is used.

Attention!
In most cases the text will be saved in RTF Format if not FormatANSI is declared. This may be dangerous when writing to normal database memos wich should be readable by other programs.

LoadFromStream
SaveToStream

WPRichText.SelText

See also

Declaration : property SelText : String

SelStart : Longint;

The SelStart property returns the starting position of the selected part of the text, with the first character in the text having a value of 0. You can use SelStart along with the SelLength property to select a portion of the text. Specify the character you want the selected text to start with by its position in the text as the value of SelStart..

SelLength : Longint;

The SelLength property returns the length (in characters) of the text which is currently selected. By using SelLength along with the SelStart property, you specify which part of the text in the control is selected. You can change the number of selected characters by changing the value of SelLength..

SelText : String ;

The SelText property contains the selected part of the control's text. You can use it to determine what the selected text is, or you can change the contents of the selected text by specifying a new string. If the selected Text is longer then 255 character, only the first 255 will be returned.

To select a text yo may use
WPRichText.Memo.SetSelPosLen(StartPos,Length), too.

WPRichText

WPRichText.SetModeControl

See also Example

Declaration : property SetModeControl : TWPSetModeControl;

SetModeControl allows to access the object wich is used to get or to set textmodes: SetModeControl

Normally you need not to use this object. It provides an interface between the Memo object (hidden in TWPRichText) to show status changes.

TWPRichText

```
{ Definition of the TWPSetModeControl }

TWPSetModeControl = class
private
  FBorder    : TBorder;
protected
  function  GetAlign   : TParAlign;
  procedure SetAlign(x : TParAlign);
  function  GetTabs : Longint;
  procedure SetTabs(x : Longint);
  function  GetBorderProp : PTBorder;
  procedure SetBorderProp(x : PTBorder);
  function  GetStyle   : WrtStyle;
  procedure SetStyle(x : WrtStyle);
  function  GetColor   : TTextColorType;
  procedure SetColor(x : TTextColorType);
  function  GetBKColor : TTextColorType;
  procedure SetBKColor(x : TTextColorType);
  function  GetFontName : TFontName;
  procedure SetFontName(x : TFontName);
  function  GetSize   : Integer;
  procedure SetSize(x : Integer);
  function  GetFirstIndent : Longint;
  function  GetRightIndent : Longint;
  function  GetLeftIndent : Longint;
  procedure SetFirstIndent(x : Longint);
  procedure SetRightIndent(x : Longint);
  procedure SetLeftIndent(x : Longint);
public
  property  Align      : TParAlign read GetAlign write SetAlign;
  property  BorderProp : PTBorder read GetBorderProp write SetBorderProp;
  property  Style      : WrtStyle read GetStyle write SetStyle;
  property  Color      : TTextColorType read GetColor write SetColor;
  property  BKColor    : TTextColorType read GetBKColor write SetBKColor;
  property  FontName   : TFontName read GetFontName write SetFontName;
  property  Size       : Integer read GetSize write SetSize;
  property  Tabs       : Longint read GetTabs write SetTabs;
  property  LeftIndent : Longint read GetLeftIndent write SetLeftIndent;
  property  RightIndent: Longint read GetRightIndent write SetRightIndent;
  property  FirstIndent: Longint read GetFirstIndent write SetFirstIndent;
public
  procedure AddStyle(x : WrtStyle);
  procedure DeleteStyle(x : WrtStyle);
  procedure ParagraphUpdate(Sender: TObject); virtual;
  procedure CharUpdate(Sender: TObject); virtual;
  procedure SetRulerOffset(NewXOffset : Longint); virtual;
```

```
public
   RTFText : TWPRTFTextIo; { _must_ be assigned }
end;

{ TWPRichText uses a TWPRichTextSetMode class, wich
  overrides some procedures: }

procedure TWPRichTextSetMode.ParagraphUpdate(Sender: TObject);
begin
   if RTFText.Header.Layout.Tabs <> Tabs then
   begin
      RTFText.Header.Layout.Tabs := Tabs;
      if assigned(FRuler) then FRuler.invalidate;
   end;
   if assigned(FToolBar) then
   begin
      case Align of
           parAlLeft      : FToolbar.SelectIcon(0,WPI_GR_ALIGN,WPI_CO_LEFT);
           parAlCenter    :
FToolbar.SelectIcon(0,WPI_GR_ALIGN,WPI_CO_CENTER);
           parAlBlock     :
FToolBar.SelectIcon(0,WPI_GR_ALIGN,WPI_CO_JUSTIFIED);
           parAlRight     : FToolBar.SelectIcon(0,WPI_GR_ALIGN,WPI_CO_RIGHT);
           end;
      end;
   if assigned(FRuler) then
FRuler.SetIndent(FirstIndent,LeftIndent,RightIndent);
end;

procedure TWPRichTextSetMode.CharUpdate(Sender: TObject);
var
  NewStyle   : WrtStyle;
  norm       : Boolean;
begin
  NewStyle := Style;
  if assigned(FToolBar) then
  begin
    norm := TRUE;
    if afsBold in NewStyle then
    begin  FToolBar.SelectIcon(0,WPI_GR_STYLE,WPI_CO_BOLD);
           norm := FALSE;
    end
    else FToolBar.deSelectIcon(0,WPI_GR_STYLE,WPI_CO_BOLD);
    if afsItalic in NewStyle then
    begin FToolBar.SelectIcon(0,WPI_GR_STYLE,WPI_CO_ITALIC);
           norm := FALSE;
```

```
    end
    else FToolBar.deSelectIcon(0,WPI_GR_STYLE,WPI_CO_ITALIC);
    if afsUnderline in NewStyle then
    begin FToolBar.SelectIcon(0,WPI_GR_STYLE,WPI_CO_UNDER);
          norm := FALSE;
    end
    else FToolBar.deSelectIcon(0,WPI_GR_STYLE,WPI_CO_UNDER);
    if afsHyperlink in NewStyle then
    begin FToolBar.SelectIcon(0,WPI_GR_STYLE,WPI_CO_HYPERLINK);
          norm := FALSE;
    end
    else FToolBar.deSelectIcon(0,WPI_GR_STYLE,WPI_CO_HYPERLINK);
    if afsStrikeOut in NewStyle then
    begin FToolBar.SelectIcon(0,WPI_GR_STYLE,WPI_CO_STRIKEOUT);
          norm := FALSE;
    end
    else FToolBar.deSelectIcon(0,WPI_GR_STYLE,WPI_CO_STRIKEOUT);
    if norm then
      FToolBar.SelectIcon(0,WPI_GR_STYLE,WPI_CO_NORMAL)
    else FToolBar.deSelectIcon(0,WPI_GR_STYLE,WPI_CO_NORMAL);

    FToolBar.UpdateSelection(wptName,FontName,0);
    FToolBar.UpdateSelection(wptSize,'',Size);
    FToolBar.UpdateSelection(wptColor,'',Color);
    FToolBar.UpdateSelection(wptBKColor,'',BKColor);
  end;
end;

procedure TWPRichTextSetMode.SetRulerOffset(NewXOffset : Longint);
begin
   if assigned(FRuler) then
   begin
      FRuler.Header  := (Parent as TWPRichText).Memo.Header;
      FRuler.XOffset := NewXOffset;
      FRuler.LeftOff := (Parent as TWPRichText).Left-FRuler.Left;
   end;
end;
```

WPRichText.StatusBar

See also

Declaration : property StatusBar : TWPStatusBarBasic;

TWPRichText will use this StatusBar to show some messages (if an stStatus string exists).

TWPRichText

WPRichText.StatusLabel

See also

Declaration : property StatusLabel : TLabel;

StatusLabel can be used for debugging porpose. The WPRichText component will use it to display messages.

You can, using WriteStatus(s:Pchar), write to this statuslabel, too. The text will be also be visible in the statusbar.

TWPRichText

ToolBar.FontSizeFrom

See also

Declaration : property FontSizeFrom : Integer;

FontSizeFrom is the smallest font size shown in the font size listbox.

TWPToolBar

ToolBar.FontSizeTo

See also

Declaration : property FontSizeTo : Integer;

FontSizeTo is the tallest font size shown in the font size listbox.

TWPToolBar

ToolBar.RtfEdit

See also

Declaration : property RtfEdit : TWPCustomRtfEdit;

RtfEdit is used by TWPToolBar to comunicate with a TWPRichText oder TDBWPRichText Object.

If you want to breake up the connection you can assign >nil< to RtfEdit. It may be neccessary to assign nil to RtfEdit inside of the OnEnetr handler of another kind of edit component. Otherwise the tollbar will send the commands to a richtext component wich has not the focus.

It may be useful in your project to use RtfEdit to find out wich component had the focus last.

TWPToolBar

ToolBar.ShowFont

See also

Declaration : property ShowFont : Boolean;

If this property is TRUE, the listbox with the fontnames will change the font for each line.

TWPToolBar

ToolBar.UpdateObject

See also

Declaration : property UpdateObject : TComponent;

The Toolbar will change the Font Style and Color of this object. The object may be a TEdit, TPanel or TMemo VCL. If you want to change an other kind of VCL you may place this on a TPanel.

TWPToolBar

ToolBar.UpdateObjectName

See also

Declaration : property UpdateObjectName : String;

The Toolbar will change the Font Style and Color of the object with the given name. The object may be a TEdit, TPanel or TMemo VCL. If you want to change an other kind of VCL you may place this on a TPanel.

TWPToolBar

ToolBar.sel_ActionIcons

See also

Declaration : property sel_ActionIcons : TWpTbIcons2;

Choose buttons for some often used actions:

SelExit
SelNew
SelOpen
SelSave
SelClose
SelPrint
SelPrintSetup

Although the TWPRichText VCL will react on selections (not SelExit and SelNew) you prbably want to add your own handler. (OnIconSelection)

TWPToolBar

ToolBar.sel_DatabaseIcons

See also

Declaration : property sel_DatabaseIcons: TWpTbIcons3;

Coose the buttons you need to navigate in the table:

SelToStart
SelNext
SelPrev
SelToEnd
SelEdit
SelAdd
SelDel
SelCancel
SelPost

A TDBWPRichText will react on the Selection of the user.

TWPToolBar

ToolBar.sel_EditIcons

See also

Declaration : property sel_EditIcons : TWpTbIcons4;

Choose the button you want to provide for editing:

SelCopy
SelCut
SelPaste
SelSelAll
SelHideSel
SelFind
SelReplace
SelSpellCheck

Please note: If the user clicked on SelSpellCheck, the TWPRichText Component will create an OnSpellCheck Event.This makes it easy for you to keep track of the focus.

TWPToolBar

ToolBar.sel_ListBoxes

See also

Declaration : property sel_ListBoxes : TWpTbListboxen;

Please choose the listboxes you want to see:

SelFontName
SelFontSize
SelFontColor
SelBackgroundColor
SelStyle, this is reserved. You may use it for any porpose.

TWPToolBar

ToolBar.sel_StatusIcons;

See also

Declaration : property sel_StatusIcons : TWpTbIcons;

Choose the writing mode buttons you want to have in the toolbar:

SelNormal
SelBold
SelItalic
SelUnder
SelHyperLink
SelStrikeOut

SelLeft
SelRight
SelBlock
SelCenter

All Selections will be handled by TWPRichText and TDBWPRichText.

TWPToolBar

TemplateName

See also

Declaration : property TemplateName : string

The given File will be loaded after the complete text was deleted. (with Clear!mClear)

You should better use a TWPRtfStorage Object.

WPRichText
LoadTemplate!mLoadTemplate
Clear!mClear
TWPRtfStorage

TextColors

See also Example

Declaration : property TextColors[Index : TWPTextColorsIndex]: TColor;

With the property TextColors you can change the color table of the RTF Text.

Note:
You can get the index of the actual textcolor in two ways:
a)
index := WPRichText1.SetModeControl.Color;
b)
var a: TAttr;
a := WPRichText1.Attr;
index := a.color and 15;

WPRichText

```
procedure Form1.ChangeTextColorClick(Sender: TObject);
var
  ind : Integer;
begin
  ind := WPRichText1.SetModeControl.Color;
  ColorDialog1.Color := WPRichText1.TextColors[ind];
  if ColorDialog1.Execute then
  begin WPRichText1.TextColors[ind] :=  ColorDialog1.Color;
        WPRichText1.Invalidate;
        WPToolBar1.Invalidate;
  end;
end;

procedure Form1.ChangeBackgroundColorClick(Sender: TObject);
var
  ind : Integer;
begin
  ind := WPRichText1.SetModeControl.BKColor;
  ColorDialog1.Color := WPRichText1.TextColors[ind];
  if ColorDialog1.Execute then
  begin WPRichText1.TextColors[ind] :=  ColorDialog1.Color;
        WPRichText1.Invalidate;
        WPToolBar1.Invalidate;
  end;
end;
```

WPRichText.WPRuler

See also

Declaration : property WPRuler : TWPRuler;

Assign if necessary the ruler the TWPRichText / TDBWPRichText component should use.

WPRichText

WPRichText.WPToolBar

See also

Declaration : property WPToolBar : TWPToolBar;

Assign if necessary the toolbar the TWPRichText / TDBWPRichText component should use.

WPRichText

WordBox

See also

Declaration : property WordBox : TWPRTFTextInput;

Don't use this property!

It is provided to enhance the compatibility to version 1.0 and 1.1 of wptools.

If you want to access the inner object at all, you should use the property Memo. Please note, only very few procedured of the inner object are documentated. The others may change in future versions.

WPRichText

Zooming

See also

Declaration : property Zooming : Integer;

With Zooming you may tell the RichText component how much the text should be enlarged or shrinked. The resolution will be changed (unlike Resizing), too.
Zooming enlarges the textfonts (instead changing the MapMode) and will reformat the complete text.

WPRichText

PrintHeader

See also

Declaration : property PrintHeader : TWPPrintEvent

The Event is created to make it possible to print on the Top Margin while printing with the procedure print.

You get TWPPrintEvents if you use the procedure print. It is quite easy to put graphics, page numbers or other information in the Header/Footer, because the EventHandler will get the x,y,w,h information where to print on the Canvas.

TWPRichText

ToolBar.Hints

See also Example

The Hints property allows you to customize the Help Hints for the buttons on the Toobar. To see the hint text you have to set ShowHint to TRUE.

Example:
WPI_Normal=Normal
WPI_Bold=Bold
WPI_Italic=Italic
WPI_Under=Underlined
WPI_Hyperlink=Hyperlink
WPI_StrikeOut=Strikeout
WPI_Left=Left aligned
WPI_Right=Right aligned
WPI_Justified=Justified text
WPI_Center=Centered text
WPI_Copy=Copy to clipboard
WPI_Cut=Cut to Clibboard
WPI_Paste=Paste from clipboard
WPI_SelAll=Select all
WPI_HideSel=Hide selection
WPI_Find=Find text
WPI_Replace=Find and replace
WPI_Spell=Call spellcheck
WPI_Exit=Terminate program
WPI_New=New file
WPI_Open=Open file
WPI_Save=Save file
WPI_Close=Close file
WPI_Print=Print Document
WPI_PrintSetup=Printer Setup
WPI_Next=Next record
WPI_Prev=Previous record
WPI_Add=Add record
WPI_Del=Delete record
WPI_Edit=Edit record
WPI_Cancel=Cancel changings
WPI_ToStart=Beginning of dataset
WPI_ToEnd=End of dataset
WPI_Post=Modify dataset

TWPToolBar