TMS TAdvRichEditor DEVELOPERS GUIDE

Jun 2015

Index

Availability

TMS TAdvRichEditor is available as VCL component for Delphi and C++Builder.

TMS TAdvRichEditor is available for Delphi XE,XE2,XE3,XE4,XE5,XE6,XE7,XE8 & C++Builder XE,XE2,XE3,XE4,XE5,XE6,XE7,XE8.

TMS TAdvRichEditor has been designed for and tested with: Windows Vista, Windows 7, Windows 8, Windows 10.

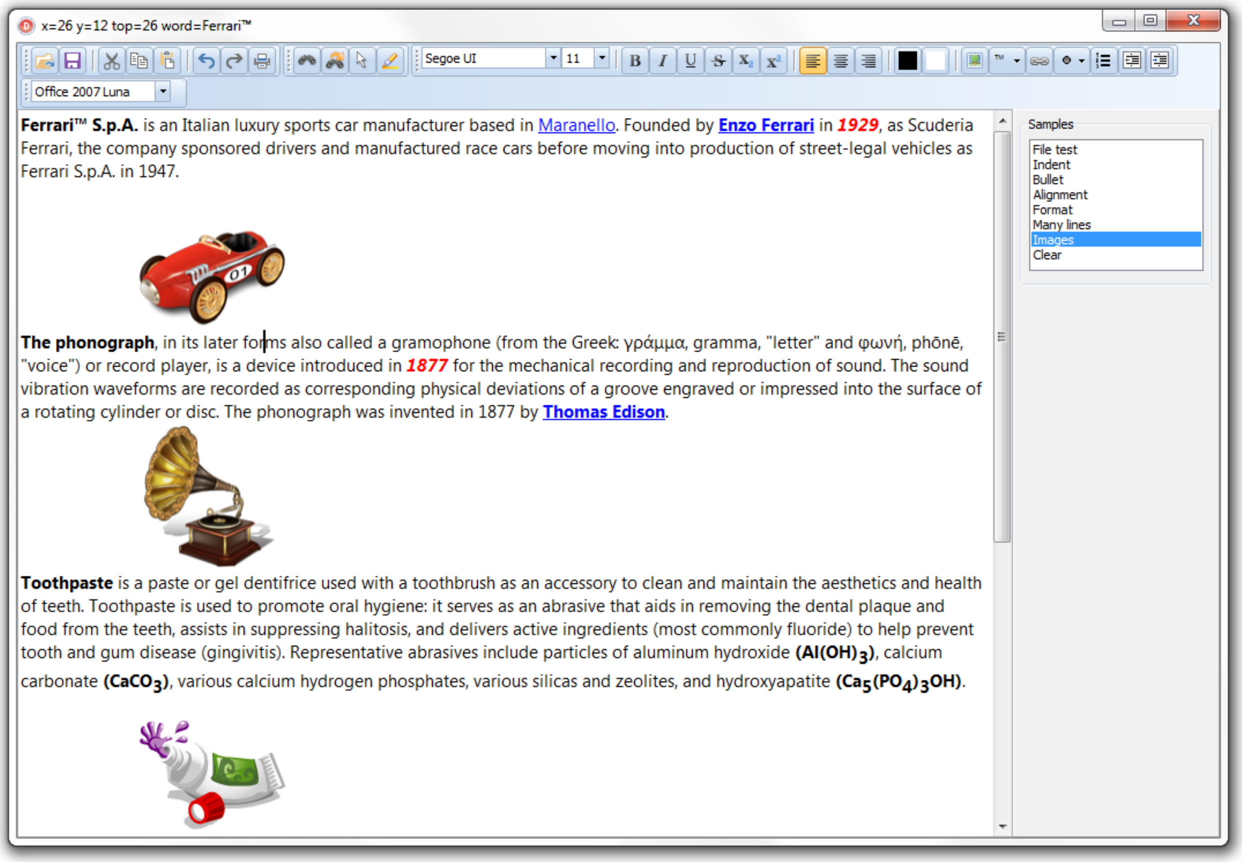
Online references

TMS software website:
http://www.tmssoftware.com

TMS TAdvRichEditor page:
http://www.tmssoftware.com/site/advricheditor.asp

Description

TAdvRichEditor is a compact light-weight wysiwyg editor for formatted text. TAdvRichEditor can include formatted text with bullets, hyperlinks, images, indenting, aligned paragraphs. It offers functions for merging, highlighting text, find & replace, undo/redo, clipboard, printing, auto-correct, emoticons.
TAdvRichEditor stores its text natively in the .RTE file format. In addition, TAdvRichEditor can import and export files in following formats: .HTML, .RTF, .TXT.
When used as part of the TMS Component Pack (http://www.tmssoftware.com/site/tmspack.asp), docking toolbars or ribbon toolbars are included to perform clipboard functions, undo/redo, formatting, paragraph alignment, inserting bullets, pictures, hyperlinks, special characters. Also available in combination with the TMS Component Pack is export to PDF and spell check.



Ferrari™ S.p.A. is an Italian luxury sports car manufacturer based in Maranello. Founded by **Enzo Ferrari** in ***1929***, as Scuderia Ferrari, the company sponsored drivers and manufactured race cars before moving into production of street-legal vehicles as Ferrari S.p.A. in 1947.

The phonograph, in its later forms also called a gramophone (from the Greek: γράμμα, gramma, "letter" and φωνή, phōnē, "voice") or record player, is a device introduced in ***1877*** for the mechanical recording and reproduction of sound. The sound vibration waveforms are recorded as corresponding physical deviations of a groove engraved or impressed into the surface of a rotating cylinder or disc. The phonograph was invented in 1877 by **Thomas Edison**.

Toothpaste is a paste or gel dentifrice used with a toothbrush as an accessory to clean and maintain the aesthetics and health of teeth. Toothpaste is used to promote oral hygiene: it serves as an abrasive that aids in removing the dental plaque and food from the teeth, assists in suppressing halitosis, and delivers active ingredients (most commonly fluoride) to help prevent tooth and gum disease (gingivitis). Representative abrasives include particles of aluminum hydroxide **($Al(OH)_3$)**, calcium carbonate **($CaCO_3$)**, various calcium hydrogen phosphates, various silicas and zeolites, and hydroxyapatite **($Ca_5(PO_4)_3OH$)**.

Organization

The core component is TAdvRichEditor. This is a standalone component that can be used as-is for WYSIWYG editing of formatted text. It comes with several toolbars (in the TMS Component Pack) that can be used to quickly setup a rich editor or its many actions can be used to create a specific user interface around the TAdvRichEditor according to your needs.

Internally the TAdvRichEditor consists of a simple DOM. This DOM is a generic list of document elements. Different types of document elements exist such as a text element, image element, linebreak element, bullet element, … Each document element has several attributes that determine the appearance in the document. While the TAdvRichEditor provides a large series of methods to add or remove elements from the DOM, it is also accessible via TAdvRichEditor.Context.Content. It is recommended though that the API used instead of direct DOM manipulation.

Getting Started

Drop a TAdvRichEditor on the form. The component with its default settings is ready for use. Entering of text can be done with default font & alignment. For ease of use, connect a TAdvRichEditorEditButtonBar and TAdvRichEditorFormatButtonBar that presents most of the built-in actions as a button bar or use the docking toolbars TAdvRichEditorEditToolBar, TAdvRichEditorFormatToolBar, TAdvRichEditorEditingToolBar, TAdvRichEditorParagraphToolBar to apply all kinds of formatting to the text without writing any code or use its ribbon equivalents for a wywiwyg editor with ribbon UI.

Properties & Events

Properties

Author	Sets the author of the document that will be persisted when saving to .RTE file format.
AutoCorrect	Contains the settings for auto-correction. TAdvRichEditor.AutoCorrect.OldValue is the string list of words to be replaced by corresponding values in the string list TAdvRichEditor.AutoCorrect.NewValue. Auto correct is enabled via setting TAdvRichEditor.AutoCorrect.Active = true. To add pairs of old/new values, use TAdvRichEditor.AutoCorrect.Add(OldValue, NewValue);
Color	Sets the default background of the TAdvRichEditor
Comments	Sets comments for the document that will be persisted when saving to .RTE file format.
Emoticons	Container of emoticon images that are used to replace common emoticon mnenomics like :), :(, …
GraphicSelection	Sets the appearance of the grips that appear when selecting graphics in the TAdvRichEditor
GraphicSelection.BorderColor	Sets the border color of graphic item grips
GraphicSelection.Color	Sets the background color of graphic item grips
GraphicSelection.Style	Selects the style between rectangular or circular for the grips
HighlightColor	Sets the background color for highlighted text in the TAdvRichEditor
HighlightTextColor	Sets the text color for highlighted text in the TAdvRichEditor
LastModifiedBy	Sets the name of the person who last modified the content of the document and this name Is persisted in the .RTE file

PictureContainer	Container of images that can be assigned to handle images inserted in the TAdvRichEditor referenced by name. PictureContainer images are inserted via TAdvRichEditor.AddNamedPicture() or TAdvRichEditor.InsertNamedPicture().
ReadOnly	When true, the content of the document cannot be altered but selection is possible
SelectionColor	Sets the background color for selection in the TAdvRichEditor
SelectionTextColor	Sets the text color for selection in the TAdvRichEditor
Tags	Sets tags for the document that will be persisted when saving to .RTE file format.
URLAuto	When set to uAuto (default), typing text starting with file://, http(s)://, ftp://, mailto: will automatically be displayed as URL in the TAdvRichEditor.
URLColor	Sets the text color for hyperlinks in the TAdvRichEditor
Version	Read-only property returning the version of the component

Events

OnCaretChanged	Event triggered whenever the caret changes in the TAdvRichEditor
OnCanSelectGraphic	Event triggered when a graphic element is clicked with allow parameter to control whether the graphic element can be selected or not
OnCanSizeGraphic	Event triggered when the mouse is over a corner of a graphic element to control whether the graphic element can be sized or not
OnClick	Event triggered when the editor is clicked
OnClickHyperlink	Event triggered when a hyperlink is clicked in the editor. The URL for the hyperlink is returned as a parameter
OnCorrectWord	Event triggered when a word is entered. The event has var parameters AWord: string and Error: Boolean. When Error is set to true, the last entered word is displayed with red error underline. When AWord is modified, this modified word is entered into the editor instead of the originally entered word.
OnDrawGraphic	Event triggered for drawing custom graphic elements in the TAdvRichEditor. This event returns the canvas and rectangle where to draw the custom graphic and an ID for the graphic element
OnEnter	Event triggered when the TAdvRIchEditor gets focus
OnEnterWord	Event triggered when one or more characters were entered before a word boundary. The event returns the word just entered
OnExit	Event triggered when the TAdvRIchEditor looses focus

OnSelectionChanged	Event triggered whenever the selection changes in the TAdvRichEditor

Methods

HasSelection: boolean;	Function returns true when there is a selection in the TAdvRichEditor
GetWordAndIndexAtCaret(var AValue: string; var AIndex: integer);	Returns the word at caret position and the index of the element containing the word
UpdateWordAndIndexAtCaret(AValue: string; AIndex: integer);	Replaces the word at document element at caret position at character index AIndex by AValue
XYToElement(X,Y: integer; var el: TREElement): boolean;	Retrieves the document element at mouse X,Y coordinates
XYToChar(X,Y: integer; el: TREElement; var CX,CY: integer): integer;	Converts the X,Y mouse coordinates to character position in the document text
XYToWord(X,Y: integer; el: TREElement): string; overload;	Returns the word and document element at mouse coordinates X,Y
XYToWord(X,Y: integer): string; overload;	Returns the word at mouse coordinates X,Y
XYToCaret(X,Y: integer); overload;	Sets the caret at mouse X,Y coordinates
XYToCaret(X,Y: single); overload;	
IsCaretInBulletList(var AType: TBulletType; var AIndex, AIndent: integer): boolean;	Returns true when the caret is within a list of bulleted items and when so, returns the bullet type, the index of the item in the list and the indent of the bulleted items
AddText(AValue: string): TTextElement; overload;	Appends text to the TAdvRichEditor and returns a text document element containing this added text
AddText(AValue: string; AColor: TColor): TTextElement; overload;	Appends text with a specific text color to the TAdvRichEditor and returns a text document element containing this added text
AddText(AValue: string; AColor: TColor; BkColor: TColor): TTextElement; overload;	Appends text with a specific text color and background color to the TAdvRichEditor and returns a text document element containing this added text
AddText(AValue: string; AFont: TFont): TTextElement; overload;	Appends text with a specific font setting to the TAdvRichEditor and returns a text document element containing this added text
AddText(AValue: string; AFontSize: integer; AFontName: string; AFontStyle: TFontStyles): TTextElement; overload;	Appends text with a specific font setting to the TAdvRichEditor and returns a text document element containing this added text
AddText(AValue: string; AAlignment: TAlignment): TTextElement; overload;	Appends text with a specific alignment to the TAdvRichEditor and returns a text document element containing this added text
AddText(AValue: string; AFontSize: integer; AFontName: string; AFontStyle: TFontStyles; AAlignment: TAlignment): TTextElement; overload;	Appends text with a specific font setting and alignment to the TAdvRichEditor and returns a text document element containing this added text
AddHyperlink(AValue, AURL: string);	Sets a hyperlink for the currently selected text in the TAdvRichEditor
AddMultiLineText(AValue: string);	Appends multiple lines of text as word-wrapped text in the TAdvRichEditor
AddLineBreak: TREElement;	Appends a linebreak to the TAdvRichEditor and returns a linebreak document element
AddBullet(AType: TBulletType = btCircle);	Appends a bullet element to the TAdvRichEditor

	and returns a bullet document element. The bullet types can be: - btSquare - btCircle - btArrow - btStar - btTick
AddImage(Picture: TPicture); overload;	Appends an image to the TAdvRichEditor and returns a graphic document element. Images of the type BMP,JPEG,GIF,PNG,ICO are supported.
AddImage(Picture: TPicture; AWidth, AHeight: integer); overload;	Appends an image with a specific width and height to the TAdvRichEditor and returns a graphic document element. Images of the type BMP,JPEG,GIF,PNG,ICO are supported.
AddImage(FileName: string); overload;	Appends an image from file to the TAdvRichEditor and returns a graphic document element
AddImage(FileName: string; AWidth, AHeight: integer); overload;	Appends an image from file with a specific width and height to the TAdvRichEditor and returns a graphic document element
AddGraphic(AWidth, AHeight: integer; AID: string);	Appends a graphical element with a specific ID to the TAdvRichEditor and returns a graphic document element. This graphical element needs to be drawn via the OnDrawGraphic event
AddNamedPicture(AWidth, AHeight: integer; AName: string);	Appends an image referenced by the unique name of the picture in an assigned PictureContainer
InsertText(Index: integer; AValue: string): TTextElement; overload;	Inserts text in the TAdvRichEditor at document element Index and returns a text document element containing this added text
InsertText(AValue: string): TTextElement; overload;	Inserts text in the TAdvRichEditor at caret position and returns a text document element containing this added text
InsertMultiLineText(AValue: string);	Inserts text in the TAdvRichEditor at caret position
InsertImage(FileName: string; AWidth: integer = 0; AHeight: integer = 0); overload;	Inserts an image with a specific width and height at caret position in the TAdvRichEditor and returns an image document element
InsertImage(Picture: TPicture; AWidth: integer = 0; AHeight: integer = 0); overload;	Inserts an image with a specific width and height at caret position in the TAdvRichEditor and returns an image document element
InsertGraphic(ID: string; AWidth, AHeight: integer);	Inserts a custom graphic element with a specific width and height at caret position in the TAdvRichEditor and returns a graphic document element
InsertNamedPicture(AName: string; AWidth, AHeight: integer);	Inserts an image referenced by the unique name of the picture in an assigned PictureContainer at caret position
InsertChar(ch: char);	Inserts a character at caret position
InsertBullet(AType: TBulletType = btCircle);	Inserts a bullet element at caret position in the TAdvRichEditor and returns a bullet document element
DeleteChar;	Deletes the character at caret position
DeleteCaretElement;	Deletes the document element where the caret is
DeleteSelection;	Deletes the selection in the TAdvRichEditor
DeleteSelected;	Deletes the selected element in case an image or graphical element is selected

SelectedText: string;	Returns the selected text
SelectWordAtXY(X,Y: integer): string;	Selects the word in the TAdvRichEditorDocument at mouse coordinates X,Y
SelectWordAtCaret: string;	Selects the word in the TAdvRichEditorDocument at caret position
WordAtXY(X,Y: integer): string;	Returns the word at X,Y mouse coordinates
WordAtCaret: string;	Returns the word at caret position
IsEmpty: boolean;	Returns true when the document is empty
Merge(NamesAndValues: TStringList);	Performs merging of mergefields with merge values contained in the stringlist
Print;	Prints the TAdvRichEditor document to the active printer
UnSelect;	Undo any selection in the document
SelectAll;	Selects all document elements in TAdvRichEditor
SaveToText(AFileName: string);	Saves the document in TAdvRichEditor as plain text
SetSelectionAttribute(AFont: TFont; AColor: TColor); overload;	Sets the font and color attribute of the seleted text
SetSelectionAttribute(AFont: TFont; AColor: TColor; BkColor: TColor); overload;	Sets the font, text color and background color attribute of the seleted text
SetSelectionAttribute(AFontName: string; AFontSize: integer; AFontStyle: TFontStyles; AColor: TColor); overload;	Sets the font and color attribute of the seleted text
SetSelectionAttribute(AFontName: string; AFontSize: integer; AFontStyle: TFontStyles; AColor, BkColor: TColor); overload;	Sets the font, text color and background color attribute of the seleted text
SetSelectionAttribute(AAlignment: TAlignment); overload;	Sets the alignment of the selected text
SetSelectionAttribute(AError: boolean); overload;	Sets the selected text with red error underlining or remove error underlining
SetSelectionColor(AColor: TColor);	Sets the text color of the selected text
SetSelectionBkColor(AColor: TColor);	Sets the background color of the selected text
SetSelectionBold(DoBold: boolean);	Sets the selected text bold or remove bold
SetSelectionItalic(DoItalic: boolean);	Sets the selected text italic or remove italic
SetSelectionUnderline(DoUnderline: boolean);	Sets the selected text underlined or remove underlined
SetSelectionStrikeOut(DoStrikeOut: boolean);	Sets the selected text strikeout or remove strikeout
SetSelectionError(DoError: boolean);	Sets the selected text with red error underlining or remove error underlining
SetSelectionSubscript(DoSubScript: boolean);	Sets the selected text subscript or remove subscript
SetSelectionSuperscript(DoSuperScript: boolean);	Sets the selected text superscript or remove superscript
SetSelectionIndent(AIndent: integer);	Sets the indent on the selected text
SetSelectionBullets(AType: TBulletType); overload;	Sets bullets for the selected text. Each line separated by a linebreak gets a bullet. AType sets the bullet type
SetSelectionHyperlink(AURL: string);	Sets a hyperlink for the text selected element in the document
SetSelectionFontName(AName: string);	Sets the font face name for the selected text
SetSelectionFontSize(ASize: integer);	Sets the font size for the selected text
SetSelectionHighlight;	Sets the selected text in highlight text / background colors

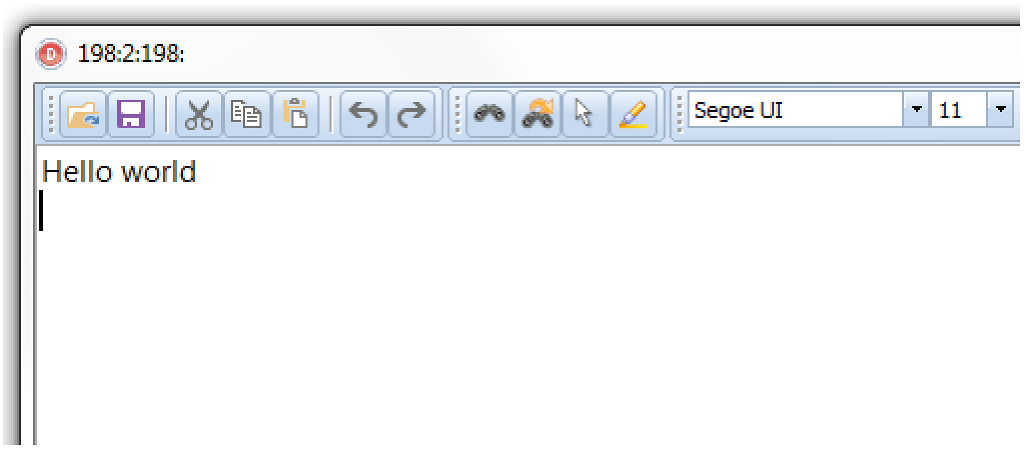
SetSelectionMergeField(AMergeName: string);	Defines a mergefield value for the selected text
IsSelectionBold: boolean;	Returns true when the selected text font style is bold
IsSelectionItalic: boolean;	Returns true when the selected text font style is italic
IsSelectionUnderline: boolean;	Returns true when the selected text font style is underline
IsSelectionStrikeOut: boolean;	Returns true when the selected text font style is strikeout
IsSelectionSubscript: boolean;	Returns true when the selected text font style is subscript
IsSelectionSuperscript: boolean;	Returns true when the selected text font style is superscript
IsSelectionLeft: boolean;	Returns true when the selected text alignment is left aligned
IsSelectionCenter: boolean;	Returns true when the selected text alignment is center aligned
IsSelectionRight: boolean;	Returns true when the selected text alignment is right aligned
GetSelectionTextColor: TColor;	Returns the text color for the selected text
GetSelectionBkColor: TColor;	Returns the background color for the selected text
GetSelectionIndent: integer;	Returns the indent of the selected text
GetSelectionFontName: string;	Returns the font face name for the selected text
GetSelectionFontSize: integer;	Returns the font size for the selected text
GetSelectionBullet: TBulletType;	Returns the bullet type used for the selected text
Clear;	Removes all elements from the document
ClearErrors;	Removes all error marking on text in the document
ClearSelection;	Clears the selection in the document
SelectText(FromChar, ALength: integer);	Selects text in the TAdvRichEditor based on character position of the text and length in characters
property Selection: TSelection read FSelection write FSelection;	Allows to get and set the selection in the TAdvRichEditor based on document elements for the selection start and selection end and character positions within the selections
property Caret: TCaret read FCaret write FCaret;	Allows to get and set the caret based on document elements and character position within the selected document element
property Selected: TREElement read FSelected write FSelected;	Get or set the selected (graphical) document element
ScrollToCaret;	Vertically scroll the TAdvRIchEditor to make the caret visible
PlainText: string;	Returns the text of the TAdvRichEditor document as plaintext
LoadFromTextFile(const FileName: string);	Loads the document from a plain text file
LoadFromTextFile(const FileName: string; Encoding: TEncoding);	Loads the document from a plain text file with optional encoding parameter
LoadFromStream(const AStream: TStream);	Load a document in the .RTE file format from stream
InsertFromStream(const AStream: TStream; f: double);	Inserts plain text from file at caret position
LoadFromFile(const FileName: string);	Load a document from the .RTE file format
SaveToFile(const FileName: string);	Save a document to the .RTE file format

SaveToStream(const AStream: TStream);	Save a document in the .RTE file format to stream
SaveSelectionToStream(const AStream: TStream);	Saves the current selected document elements in .RTE file format to stream
FindFirst(AText: string; MatchCase: boolean = false): boolean;	Finds the first occurrence of text from the document origin
FindNext: boolean;	Finds the next occurrence of text from the position of the last find operation
ReplaceFirst(AText, AReplacement: string; MatchCase: boolean = false): boolean;	Replaces the first occurrence of text from the document origin
ReplaceNext: boolean;	Replaces the next occurrence of text from the position of the last find operation
Highlight(AText: string; MatchCase: boolean = false): boolean;	Highlight the text in the document with or without case sensitivity in the document
UnHighlight;	Undo any previous highlight
CanUnindent: boolean;	Returns true when the selection in the document is indented (and thsu can be unindented)
CanUndo: boolean;	Returns true when an Undo operation is possible
CanRedo: boolean;	Returns true when a Redo operation is possible
Undo;	Performs Undo
Redo;	Performs Redo
BeginUpdate;	Use to block updates when doing many programmatic manipulations in the TAdvRichEditor
EndUpdate;	Use to block updates when doing many programmatic manipulations in the TAdvRichEditor

Programmatic access to the document

Text can be inserted in TAdvRichEditor in various ways. To start with call:

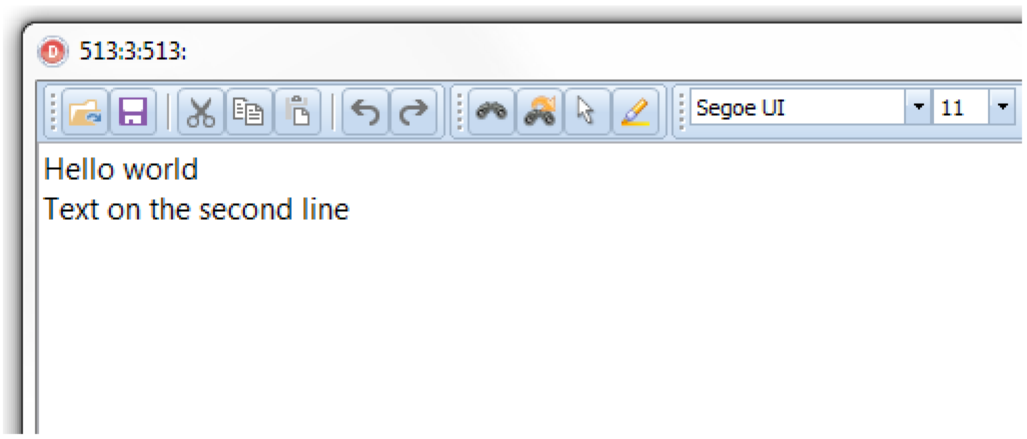
```
AdvRichEditor.AddText('Hello world');
```



Hello world

Add text on the next line with:

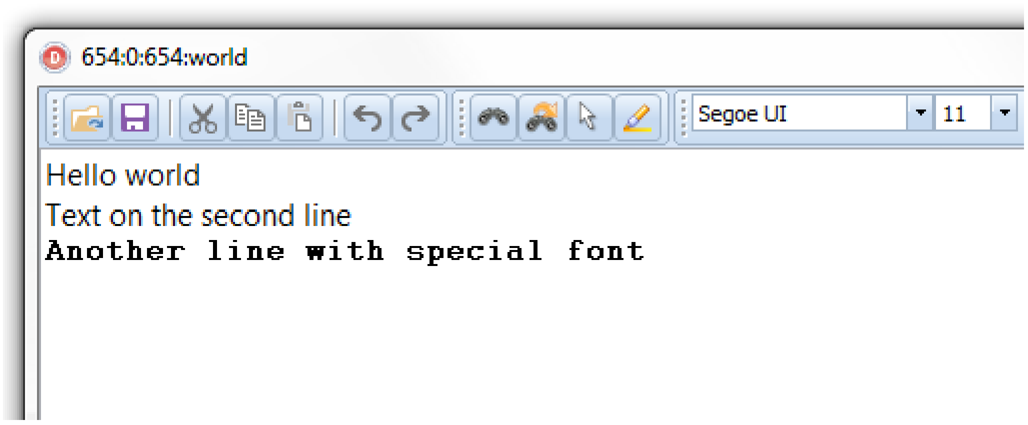
```
AdvRichEditor.AddLineBreak;
AdvRichEditor.AddText('Text on the second line');
```



Hello world
Text on the second line

To add text with a different font than default font, use:

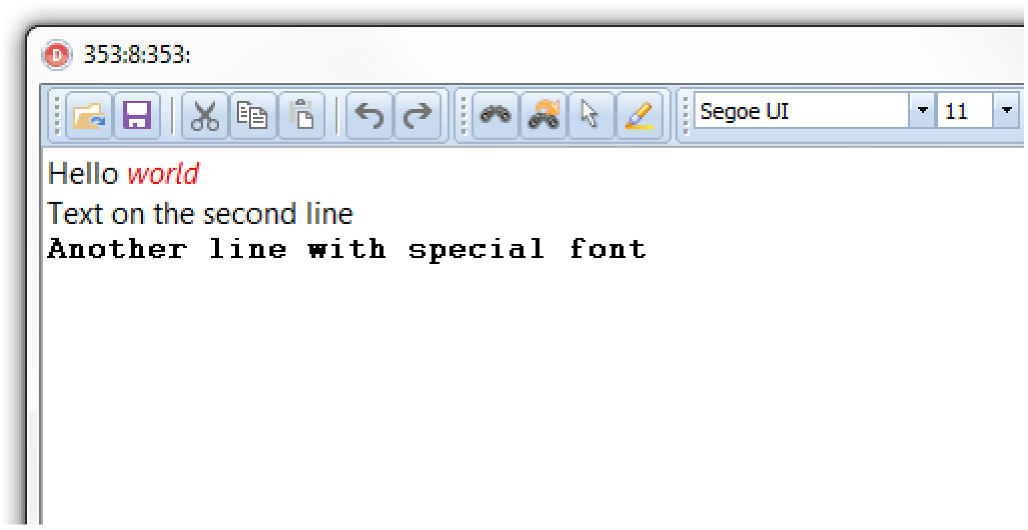
```
AdvRichEditor.AddLineBreak;
AdvRichEditor1.AddText('Another line with special
font',12,'Courier',[fsBold]);
```



Hello world
Text on the second line
Another line with special font

To change attributes of text in the TAdvRichEditor, perform a selection based on index of the text and length. For example, to change the color of "world" on the first line, set a selection from character 6 for 5 characters (character index starts at zero) and set an attribute for the selection followed by remove the selection itself:

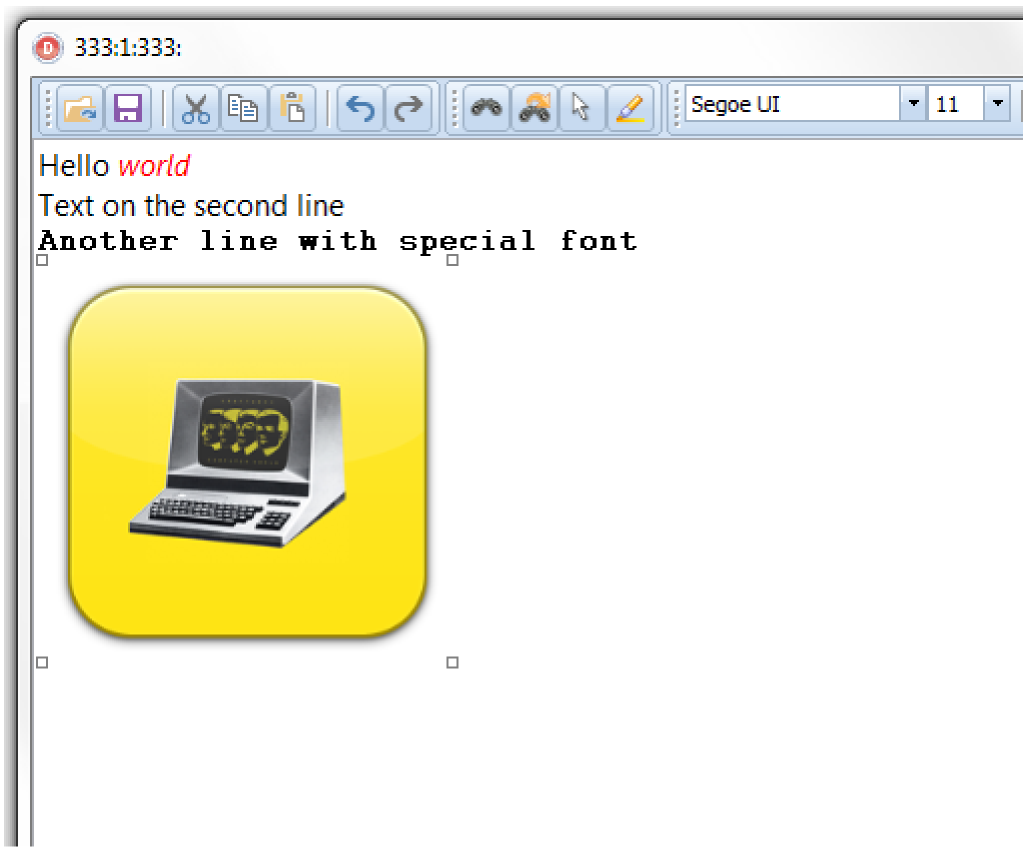
```
AdvRichEditor1.SelectText(6,5);
AdvRichEditor1.SetSelectionColor(clRed);
AdvRichEditor1.SetSelectionItalic(True);
AdvRichEditor1.ClearSelection;
```



Hello *world*
Text on the second line
Another line with special font

To add images to the TAdvRichEditor, use:

```
AdvRichEditor1.AddImage('.\sample.png');
```



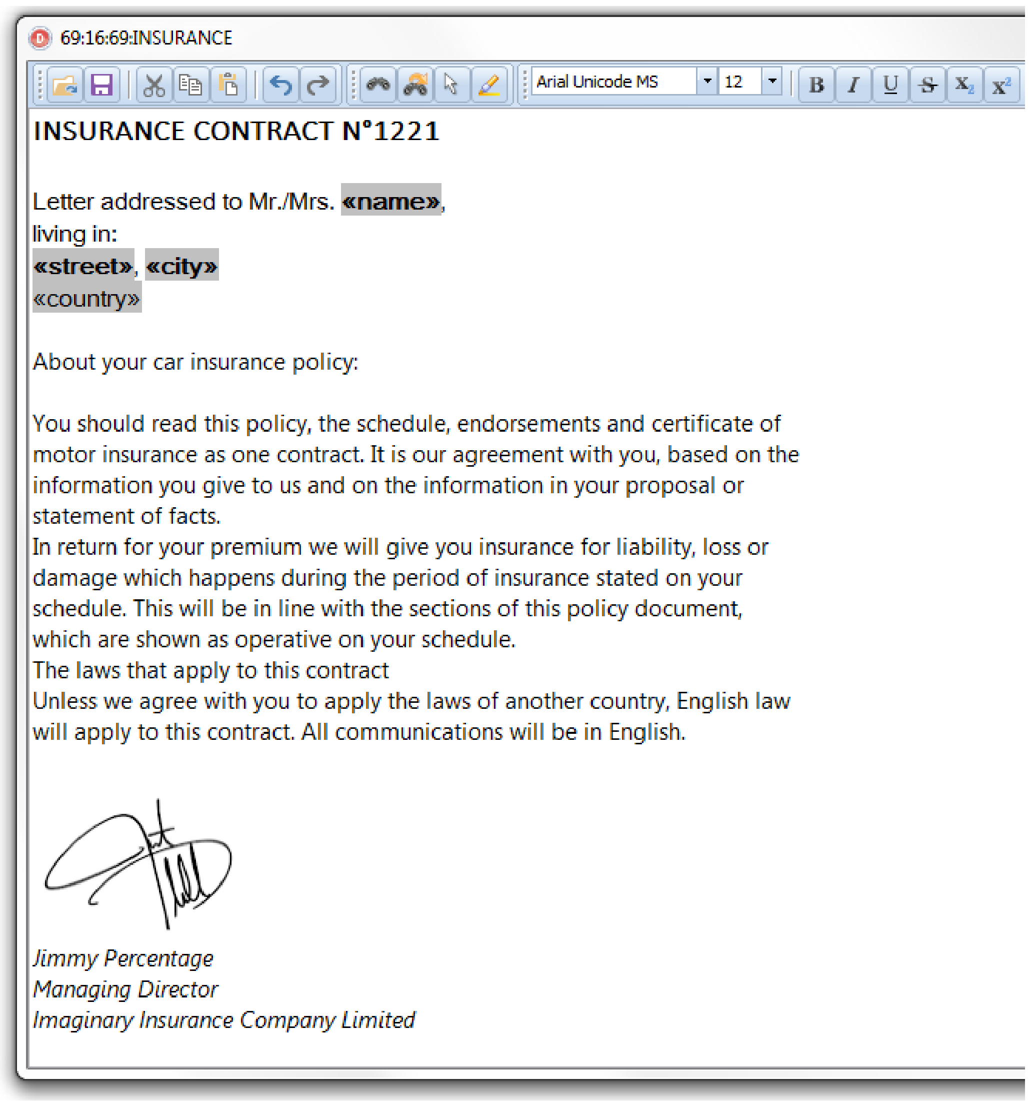
Hello *world*
Text on the second line
Another line with special font

Using merge fields

Via merge fields, specific places in the document can be quickly replaced during a merge operation. To perform merging, first insert merge fields in the document. Merge fields are pieces of text that get a merge field name. These pieces of text are displayed between brackets «» and with a gray background. To set a piece of text as merge field, select the text and call AdvRichEditor.SetSelectionMergeField('MergeFieldName');

Assume that following merge field names exist in the TAdvRichEditor document:

'Name'
'Street'
'City'
'Country'



INSURANCE CONTRACT N°1221

Letter addressed to Mr./Mrs. **«name»**,
living in:
«street», **«city»**
«country»

About your car insurance policy:

You should read this policy, the schedule, endorsements and certificate of motor insurance as one contract. It is our agreement with you, based on the information you give to us and on the information in your proposal or statement of facts.
In return for your premium we will give you insurance for liability, loss or damage which happens during the period of insurance stated on your schedule. This will be in line with the sections of this policy document, which are shown as operative on your schedule.
The laws that apply to this contract
Unless we agree with you to apply the laws of another country, English law will apply to this contract. All communications will be in English.

[illegible]

Jimmy Percentage
Managing Director
Imaginary Insurance Company Limited

then a merge operation can be done in the following way:

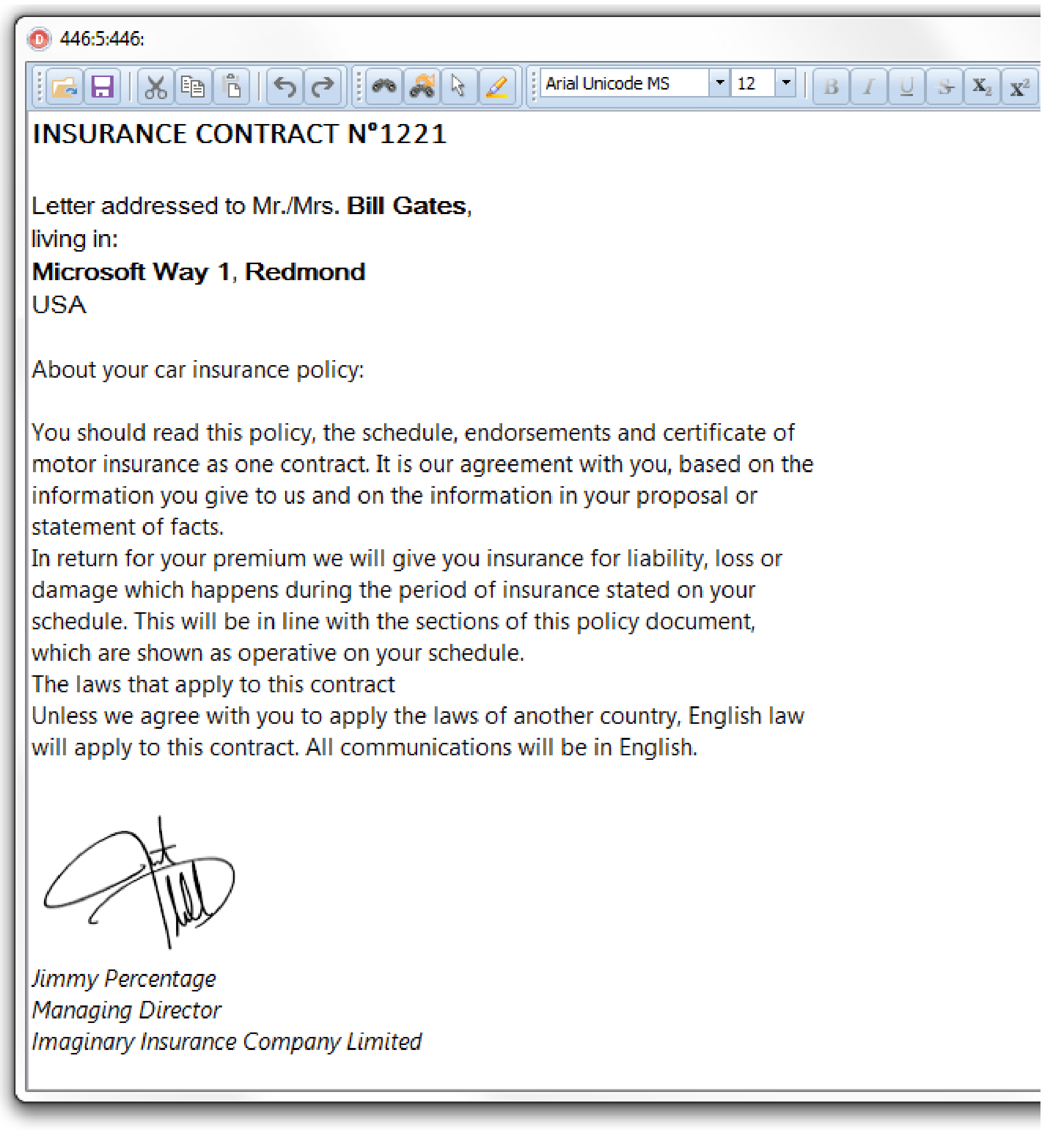
```
var
  sl: TStringList;
```

```
sl := TStringList.Create;
sl.Add('Name=Bill Gates');
sl.Add('Street=Microsoft Way 1');
sl.Add('City=Redmond');
sl.Add('Country=USA');

AdvRichEditor1.Merge(sl);

sl.Free;
```

This will replace the merge fields Name, Street, City, Country with the values 'Bill Gates', 'Microsoft Way 1', 'Redmond', 'USA' specifically.



INSURANCE CONTRACT N°1221

Letter addressed to Mr./Mrs. **Bill Gates**,
living in:
Microsoft Way 1, **Redmond**
USA

About your car insurance policy:

You should read this policy, the schedule, endorsements and certificate of motor insurance as one contract. It is our agreement with you, based on the information you give to us and on the information in your proposal or statement of facts.
In return for your premium we will give you insurance for liability, loss or damage which happens during the period of insurance stated on your schedule. This will be in line with the sections of this policy document, which are shown as operative on your schedule.
The laws that apply to this contract
Unless we agree with you to apply the laws of another country, English law will apply to this contract. All communications will be in English.

[illegible]

Jimmy Percentage
Managing Director
Imaginary Insurance Company Limited

It is also possible to replace merge fields by pictures, i.e. insert pictures dynamically during a merge operation.

To do this, set a merge fieldname just like for text but using following construct for the mergelist:

Assume that in the previous example we want to add a picture of the person in the document, this would become:

‘Photo’
‘Name’
‘Street’
‘City’
‘Country’

A merge operation is be done in the following way:

```
var
  sl: TStringList;
  pic: TPicture;

pic := TPicture.Create;
pic.LoadFromFile(‘billgates.jpg’);

sl := TStringList.Create;
sl.AddObject(‘Photo=’,pic);
sl.Add(‘Name=Bill Gates’);
sl.Add(‘Street=Microsoft Way 1’);
sl.Add(‘City=Redmond’);
sl.Add(‘Country=USA’);

AdvRichEditor1.Merge(sl);

sl.Free;
pic.Free;
```

To undo the merge operation (and have the document ready for a new merge operation), simply call AdvRichEditor1.UnMerge; after the merge operation.

Using accompanying toolbars

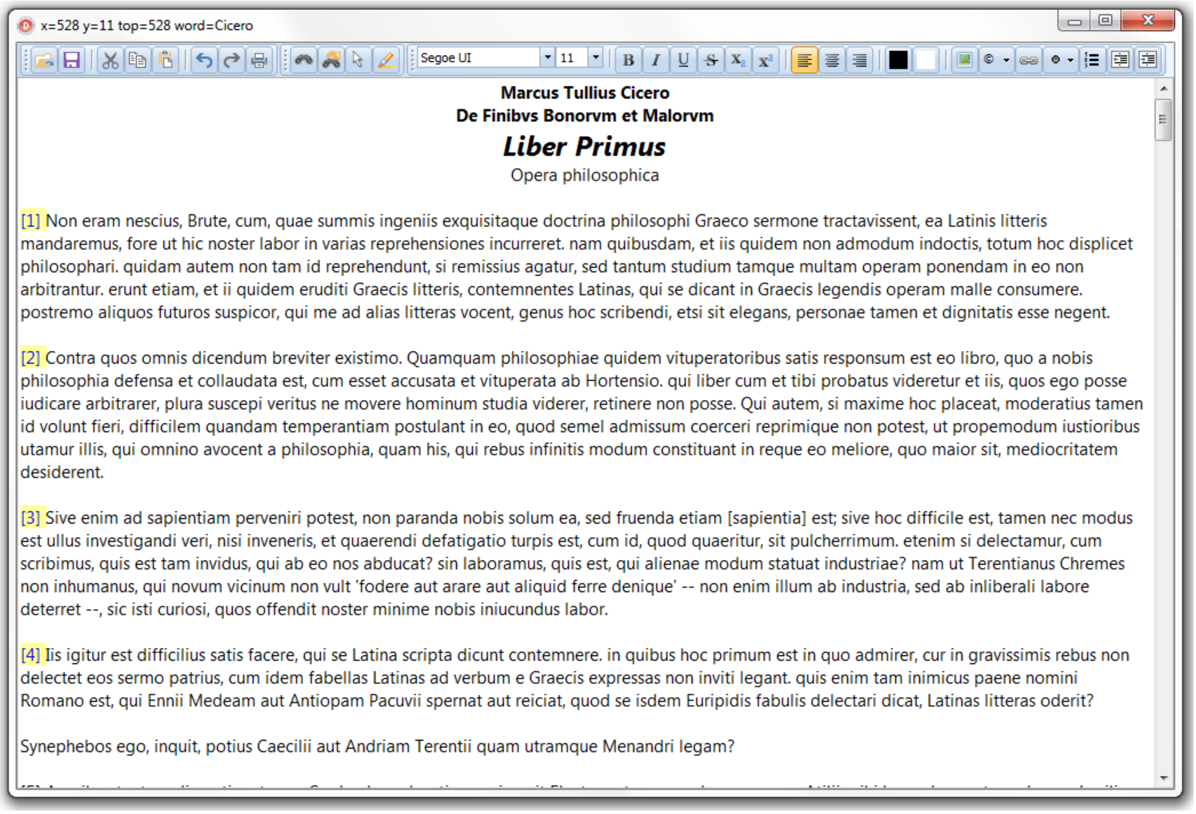
TAdvRichEditor comes with several ready-to-use toolbars that enable to quickly create user-interfaces for manipulating the formatted text without writing code.

The toolbars come in 3 categories:

- Simple button bars that only rely on standard VCL controls
- A docking toolbar set that builds upon the docking toolbars found in TMS Advanced Toolbars & Menus (see http://www.tmssoftware.com/site/advtoolbars.asp)
- Ribbon toolbars that also build upon TMS Advanced Toolbars & Menus

The offered toolbars all make internally heavily use of actions to achieve their functionality. Some functions in the toolbars still need a reference to the TAdvRichEditor instance that is being worked with and therefore, it is needed to set ToolBar.RichEditor to the instance of this TAdvRichEditor.

Sample for a UI made with docking toolbars:



Marcus Tullius Cicero
De Finibvs Bonorvm et Malorvm
Liber Primus
Opera philosophica

[1] Non eram nescius, Brute, cum, quae summis ingeniis exquisitaque doctrina philosophi Graeco sermone tractavissent, ea Latinis litteris mandaremus, fore ut hic noster labor in varias reprehensiones incurreret. nam quibusdam, et iis quidem non admodum indoctis, totum hoc displicet philosophari. quidam autem non tam id reprehendunt, si remissius agatur, sed tantum studium tamque multam operam ponendam in eo non arbitrantur. erunt etiam, et ii quidem eruditi Graecis litteris, contemnentes Latinas, qui se dicant in Graecis legendis operam malle consumere. postremo aliquos futuros suspicor, qui me ad alias litteras vocent, genus hoc scribendi, etsi sit elegans, personae tamen et dignitatis esse negent.

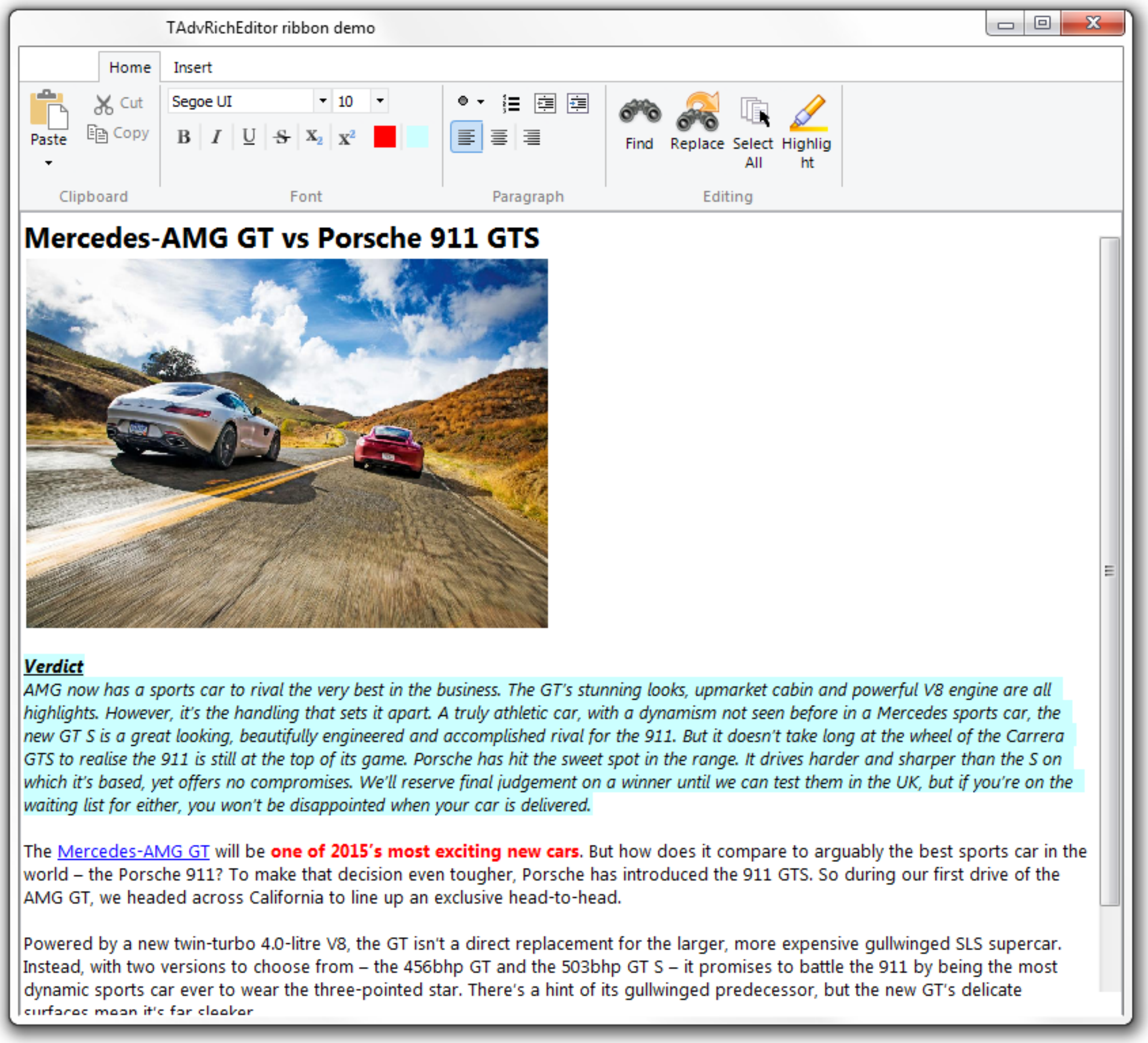
[2] Contra quos omnis dicendum breviter existimo. Quamquam philosophiae quidem vituperatoribus satis responsum est eo libro, quo a nobis philosophia defensa et collaudata est, cum esset accusata et vituperata ab Hortensio. qui liber cum et tibi probatus videretur et iis, quos ego posse iudicare arbitrarer, plura suscepi veritus ne movere hominum studia viderer, retinere non posse. Qui autem, si maxime hoc placeat, moderatius tamen id volunt fieri, difficilem quandam temperantiam postulant in eo, quod semel admissum coerceri reprimique non potest, ut propemodum iustioribus utamur illis, qui omnino avocent a philosophia, quam his, qui rebus infinitis modum constituant in reque eo meliore, quo maior sit, mediocritatem desiderent.

[3] Sive enim ad sapientiam perveniri potest, non paranda nobis solum ea, sed fruenda etiam [sapientia] est; sive hoc difficile est, tamen nec modus est ullus investigandi veri, nisi inveneris, et quaerendi defatigatio turpis est, cum id, quod quaeritur, sit pulcherrimum. etenim si delectamur, cum scribimus, quis est tam invidus, qui ab eo nos abducat? sin laboramus, quis est, qui alienae modum statuat industriae? nam ut Terentianus Chremes non inhumanus, qui novum vicinum non vult 'fodere aut arare aut aliquid ferre denique' -- non enim illum ab industria, sed ab inliberali labore deterret --, sic isti curiosi, quos offendit noster minime nobis iniucundus labor.

[4] Iis igitur est difficilius satis facere, qui se Latina scripta dicunt contemnere. in quibus hoc primum est in quo admirer, cur in gravissimis rebus non delectet eos sermo patrius, cum idem fabellas Latinas ad verbum e Graecis expressas non inviti legant. quis enim tam inimicus paene nomini Romano est, qui Ennii Medeam aut Antiopam Pacuvii spernat aut reiciat, quod se isdem Euripidis fabulis delectari dicat, Latinas litteras oderit?

Synephebos ego, inquit, potius Caecilii aut Andriam Terentii quam utramque Menandri legam?

Sample for a UI made with ribbon:



Mercedes-AMG GT vs Porsche 911 GTS

Verdict

AMG now has a sports car to rival the very best in the business. The GT's stunning looks, upmarket cabin and powerful V8 engine are all highlights. However, it's the handling that sets it apart. A truly athletic car, with a dynamism not seen before in a Mercedes sports car, the new GT S is a great looking, beautifully engineered and accomplished rival for the 911. But it doesn't take long at the wheel of the Carrera GTS to realise the 911 is still at the top of its game. Porsche has hit the sweet spot in the range. It drives harder and sharper than the S on which it's based, yet offers no compromises. We'll reserve final judgement on a winner until we can test them in the UK, but if you're on the waiting list for either, you won't be disappointed when your car is delivered.

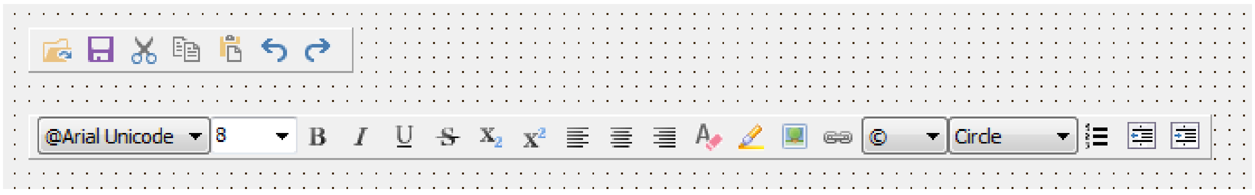
The Mercedes-AMG GT will be **one of 2015's most exciting new cars**. But how does it compare to arguably the best sports car in the world – the Porsche 911? To make that decision even tougher, Porsche has introduced the 911 GTS. So during our first drive of the AMG GT, we headed across California to line up an exclusive head-to-head.

Powered by a new twin-turbo 4.0-litre V8, the GT isn't a direct replacement for the larger, more expensive gullwinged SLS supercar. Instead, with two versions to choose from – the 456bhp GT and the 503bhp GT S – it promises to battle the 911 by being the most dynamic sports car ever to wear the three-pointed star. There's a hint of its gullwinged predecessor, but the new GT's delicate surfaces mean it's far sleeker

To start using the toolbars, simple drop one of the needed buttonbar on the form or toolbars on either a TAdvDockPanel or TAdvOfficePage.

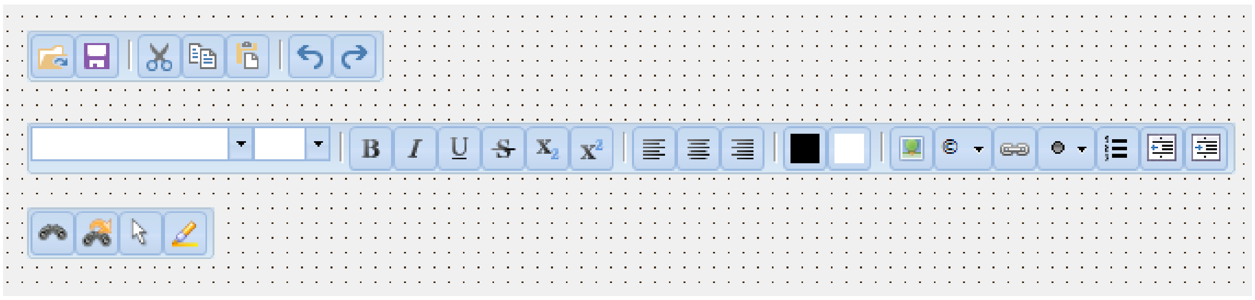
TAdvRichEditorEditButtonBar,TAdvRichEditorFormatButtonBar

These are two button bars, built using standard VCL Panels & speedbuttons. The TAdvRichEditorEditButtonBar gives access to file open/save functions, clipboard functions and Undo/Redo. The TAdvRichEditorFormatButtonBar offers all control over font, colors, alignment, indenting, bullets as well as inserting images, hyperlinks and special characters.



TAdvRichEditorEditToolBar, TAdvRichEditorFormatToolBar, TAdvRichEditorEditingToolbar

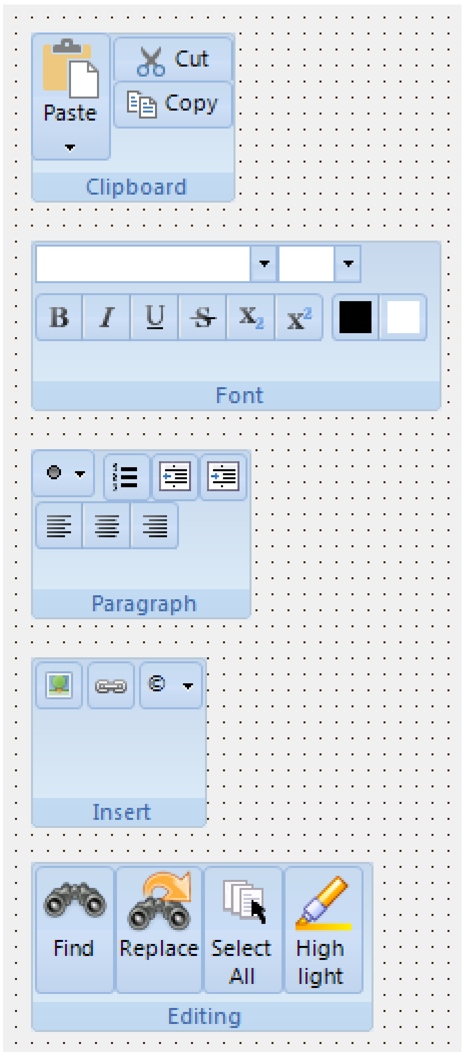
These are three toolbars designed to be used in combination with a TAdvDockPanel. The toolbars are divided in functions for Open/Save/Clipboard/Undo/Redo with the TAdvRichEditorEditToolBar, changing font characteristics, alignment, bullets, indents, colors and inserting images, hyperlinks, special characters with the TAdvRichEditorFormatToolbar and finally, Find & Replace, highlight and Select-All with the TAdvRichEditorEditingToolbar.



TAdvRichEditorClipboardRibbonToolBar, TAdvRichEditorFormatRibbonToolBar, TAdvRichEditorParagraphRibbonToolBar, TAdvRichEditorInsertRibbonToolBar, TAdvRichEditorEditingRibbonToolBar

For use with a ribbon user-interface, the different functions for the TAdvRichEditor were divided in 5 parts:

TAdvRichEditorClipboardRibbonToolBar: clipboard functions
TAdvRichEditorFormatRibbonToolBar: font formatting functions
TAdvRichEditorParagraphRibbonToolBar: alignment, indenting, bullet functions
TAdvRichEditorInsertRibbonToolBar: inserting images, hyperlink, special characters
TAdvRichEditorEditingRibbonToolBar: find & replace, select-all and highlight



Importing & exporting in rich text

TAdvRichEditor comes with a component to allow to import or export its content in rich text (.RTF) files.
Performing such export or import is easy. Drop a TAdvRichEditorRTFIO component on the form and connect the TAdvRichEditor to this non-visual component's RichEditor property.

Export

Simply call:

AdvRichEditorRTFIO.Save(FileName);

Import

Simply call:

AdvRichEditorRTFIO.Load(FileName);

Importing & exporting in HTML format

TAdvRichEditor comes with a component to allow to export its content in HTML (.HTML) files. It is also possible to import from files that use a HTML subset (mini HTML) described here: http://www.tmssoftware.com/site/minihtml.asp

Performing such export or import is easy. Drop a TAdvRichEditorHTMLIO component on the form and connect the TAdvRichEditor to this non-visual component's RichEditor property.

Export

Simply call:

AdvRichEditorHTMLIO.Save(FileName);

Notice that for HTML export, the default behaviour is that all images used in the document are exported as separate linked image files in the same folder where the .HTML file is generated. If it is preferred that images are generated in a different folder, use the 2nd default parameter ImagePath:

AdvRichEditorHTMLIO.Save(FileName, ImagePath);

Import

This is limited to mini HTML files and import is done via the non-visual component TAdvRichEditorMiniHTMLIO. In the same way as TAdvRichEditorHTMLIO, assign the TAdvRichEditor instance via TAdvRichEditorMiniHTMLIO.RichEditor. The component provides the following overloads to import from HTML:

```
    procedure Load(HtmlValue: string; const Images: TCustomImageList; const
Pictures: TGDIPPictureContainer = nil); overload;
    procedure Load(FileName: string); overload;
procedure Load(AStream: TStream); overload;
```

This way, it can import from a simple HTML formatted string, a file with HTML formatted text or a stream. In the case of loading from a HTML formatting string, 2 extra parameters Images & Pictures can be used as containers for referenced images in the HTML formatted string.

Finally, one more helper method is available in TAdvRichEditorMiniHTMLIO:

```
procedure Insert(HtmlValue: string);
```

This inserts the formatted text from a HTML formatted string at caret position in the TAdvRichEditor.

Exporting to PDF

The TMS Component Pack also contains a component for exporting the TAdvRichEditor content to PDF file.
Drop a TAdvRichEditorPDFIO component on the form and connect the TAdvRichEditor to this non-visual component's RichEditor property.

Then simply call:

AdvRichEditorPDFIO.Save(FileName);

TAdvRichEditorPDFIO comes with settings for header and footer as well as metadata. Header and footer can as such be optionally generated for the PDF file independently from the TAdvRichEditor content.

Import or export to mini-HTML

With the component TAdvRichEditor.MiniHTMLIO, it is possible to read or write the contents of the TAdvRichEditor in mini-HTML format. Mini-HTML is a subset of HTML and is described at:
http://www.tmssoftware.com/site/minihtml.asp

To use TAdvRichEditorMiniHTML to read or write its contents in HTML, drop TAdvRichEditorMiniHTML on the form and connect the TAdvRichEditor instance to TAdvRichEditorMiniHTMLIO.RichEditor.

Call TAdvRichEditorMiniHTMLIO.Load(FileName: string) to load the content from a HTML file.

Call TAdvRichEditorMiniHTMLIO.Save(FileName: string) to save the content to a HTML file.

In addition to saving to file, it is also possible to save to a stream or get the content as HTML:

TAdvRichEditorMiniHTMLIO.Save(AStream: TStream) : saves the content in HTML format to stream

TAdvRichEditorMiniHTMLIO.AsString: string : returns the content in HTML format as string

In addition to loading from file, it is also possible to get the content from a stream or a HTML formatted string:

TAdvRichEditorMiniHTMLIO.Load(AStream: TStream);

Loads the content from a stream containing the HTML formatted text.

TAdvRichEditorMiniHTMLIO.Load(HtmlValue: string; const Images: TCustomImageList; const Pictures: TPictureContainer = nil);

HTMLValue contains the content as HTML formatted string. Optionally, for passing pictures, an imagelist or picturecontainer can be used in case the HTML formatted string references pictures in an imagelist or picturecontainer.

TAdvRichEditor actions

TAdvRichEditor also registered various actions that can be used to quickly hookup visual controls to perform actions on the TAdvRichEditor.

Currently following actions are registered and available from the action manager in the IDE:

TAdvRichEditorClear	Clear document
TAdvRichEditorCut	Cut selection to clipboard
TAdvRichEditorCopy	Copy selection to clipboard
TAdvRichEditorPaste	Paste text or image from clipboard
TAdvRichEditorSelectAll	Selects all text in document
TAdvRichEditorAlignRight	Align paragraph right
TAdvRichEditorAlignCenter	Align paragraph centered
TAdvRichEditorAlignLeft	Align paragraph left
TAdvRichEditorBold	Toggle bold font style on selection
TAdvRichEditorItalic	Toggle italic font style on selection
TAdvRichEditorUnderline	Toggle underline font style on selection
TAdvRichEditorStrikeOut	Toggle strikeout font style on selection
TAdvRichEditorSubScript	Toggle subscript font style on selection
TAdvRichEditorSuperScript	Toggle superscript font style on selection
TAdvRichEditorTextColor	Sets text color of selected text
TAdvRichEditorFontName	Sets font face name of selected text
TAdvRichEditorFontSize	Sets font size of selected text
TAdvRichEditorBulletType	Sets the bullet type of the selected text
TAdvRichEditorNumberedBulletType	Sets numbered bullets on selected text
TAdvRichEditorColor	Sets background color on selected text
TAdvRichEditorIndent	Indent selected text
TAdvRichEditorUnIndent	Unindent selected text
TAdvRichEditorUndo	Perform Undo
TAdvRichEditorRedo	Perform Redo

TDBAdvRichEditor

TDBAdvRichEditor is a DB-aware version of TAdvRichEditor. It allows to connect the rich editor directly to a database that stores the content of the editor in a blob field.
To start using the TDBAdvRichEditor, drop a dataset on the form and connect it via a datasource to the TDBAdvRichEditor.DataSource property. Set the blob field where the content is stored via TDBAdvRichEditor.DataField.

In addition to enabling to hookup the content of the rich editor to a dataset, the TDBAdvRichEditor can also perform merging with another dataset. To do this, drop a merge dataset on the form and connect it via a datasource to TDBAdvRichEditor.MergeSource. When executing TDBAdvRichEditor.Merge, the rich editor will now try to find the value for the merge fields in the dataset.

Example:

When the merge dataset contains the DB fields:

NAME: VARCHAR(25)
PRENAME: VARCHAR(25)
CITY: VARCHAR(25)
PICTURE: BLOB

then, on the position in the text where this DB field values should be inserted, add merge fields with the name set to the DB fieldname, i.e. use DBAdvRichEditor.SetSelectionMergeField(DBFieldName) for each of the fields.

When TDBAdvRichEditor.Merge is now called, the field data (text / images) from the current record in the merge dataset will now be set at these positions in the rich editor.

Spell check with TAdvRichEditor

Not available in the stand-alone TAdvRichEditor product but as part of the TMS Component Pack, it comes with a spell check engine and several UI components to perform spell check & auto-correct either while typing in the TAdvRichEditor or statically on the document.
To use spell check with TAdvRichEditor, simply drop the non-visual component TAdvRichEditorSpellCheck on the form and connect the TAdvRichEditor instance to TAdvRichEditorSpellCheck.RichEditor.

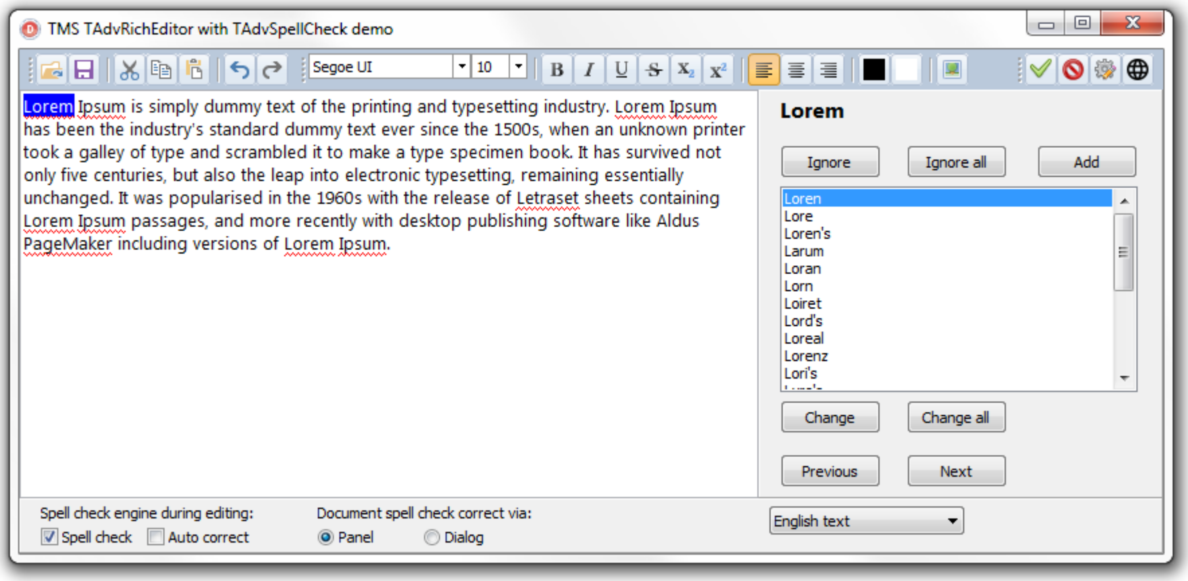
With the option TAdvRichEditorSpellCheck.SpellCheckAction, it can be controlled what action is performed during typing.

SpellCheckAction:

spcNone: while typing, no action is performed
spcMarkError: when a misspelled word is typed, it is marked with red underline
spcAutoCorrect: when a misspelled word is typed, it is automatically replaced by the first matching word

Other than performing a spell check while typing, it is possible to perform a spell check on the entire document.

The spell check on the entire document is initiated with TAdvRichEditorSpellCheck.CheckDocument:



Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Spell check engine during editing: [✓ Spell check] Auto correct
Document spell check correct via: [✓ Panel] Dialog
English text

This performs an asynchronous spell check of the document. When the spell check is finished, the event OnRequestsProcessed is triggered.

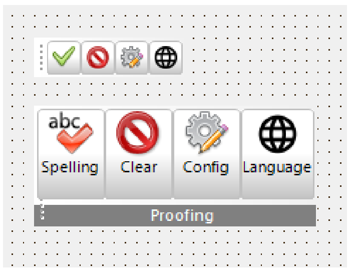
In addition to the core spell check engine for TAdvRichEditor, two additional user interface components are available: TAdvRichEditorSpellCheckDialog and TAdvRichEditorSpellCheckPanel. These components can be dropped on the form. Connect the spell check engine and TAdvRichEditor instance to the TAdvRichEditorSpellCheckPanel.SpellCheck and TAdvRichEditorSpellCheckPanel.RichEditor properties respectively. This allows to automatically interact with the correction of a spell-checked document without any code needed except the initialization when the spell check is complete. This is commonly done with:

```
procedure TForm1.AdvRichEditorSpellCheck1RequestsProcessed(Sender: TObject;
  Context: TProcessRequestContext);
begin
  AdvRichEditor1.SelectError(esFirst);
  AdvRichEditorSpellCheckPanel1.DoUpdate;
end;
```

This code selects the first incorrect word in the TAdvRichEditor instance and then instructs the panel to initialize itself with the suggestions list for this first misspelled word.

Built-in proofing docking toolbar and proofing ribbon toolbar

TAdvRichEditor also comes with ready-to-use docking & ribbon proofing toolbars:



The toolbar features four buttons:

- Perform spell check of the document
- Clear all marked spell check errors
- Configure the spell check engine
- Choose the language

Via the Options property, any of these toolbar buttons can be turned on or turned off.