tmssoftware·com

VCL

TMS TAdvMemo - TDBAdvMemo
DEVELOPERS GUIDE

Jul 2017

Copyright © 2001 – 2017 by tmssoftware.com bvba
Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

1

Table of contents

Introduction

The TMS TAdvMemo is a highly configurable syntax highlighting memo control.

The TMS TAdvMemo has features like code-folding, auto completion, breakpoints and bookmarks and markers support, and holds a source code container.

In this document you will find an overview of the TAdvMemo component and its features, code snippets to quickly start using the component and overviews of properties, methods and events.

These properties, methods and events also apply to the TDBAdvDemo Control.

Availability

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

TMS TAdvMemo is available for Delphi 7, 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin, 10.2 Tokyo and C++Builder 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, 10.1 Berlin, 10.2 Tokyo (Prof/Enterprise/Architect)

TMS TAdvMemo has been designed for and tested with: Vista, Windows 7, Windows 8, Windows 10in Win32 applications , C++Builder XE2 or newer versions

List of included components

TAdvMemo: Is the core memo component, and supplies syntax checking for a large number of programming environments.

TDBAdvMemo: Is the data aware version of TAdvMemo.

TAdvCodeList: Gives the possibility to create a list with predefined program code blocks that can be dragged & dropped into the TAdvMemo canvas.

TAdvMemoSource: Allows connecting different text files to one TAdvMemo control. This can be used to optimize memory usage in multi document programs.

TAdvMemoFindDialog: Allows searching for a certain (sub-) string in the TAdvMemo control.

TAdvMemoFindReplaceDialog: Allows searching for a certain (sub-) string in the TAdvMemo control, and replace it with another text string.

TAdvFindDialog: Extended find dialog with specifiers for expressions in search, auto history of search strings.

TAdvReplaceDialog: Extended find & replace dialog with specifiers for expressions in search, auto history of search & replace strings.

TAdvMemoCapitalChecker: Is an extension on the standard TAdvMemo that allows checking for capital letters at the start of a sentence. This feature is useful when creating a standard text editor.

TAdvMemoStylerManager: Allows managing of an existing MemoStyler (or even creating a new one). MemoStylers provide predefined color-coding of user-defined code syntax.

TAdvBasicMemoStyler: Is the MemoStyler file for MS Visual Basic files syntax checking.

TAdvCSharpMemoStyler: Is the MemoStyler for C# files syntax checking.

TAdvCSSMemoStyler: Is the MemoStyler for CSS files syntax checking.
TAdvEmoticonMemoStyler: Is the MemoStyler for Emoticon files syntax checking.

TAdvHTMLMemoStyler: Is the MemoStyler for HTML files syntax checking.

TAdvINIMemoStyler: Is the MemoStyler for INI files syntax checking.

TAdvJSMemoStyler: Is the MemoStyler for JavaScript files syntax checking.

TAdvPascalMemoStyler: Is the MemoStyler for Pascal files syntax checking.

TAdvPerlMemoStyler: Is the MemoStyler for Perl files syntax checking.

TAdvPHPMemoStyler: Is the MemoStyler for PHP files syntax checking.

TAdvPythonMemoStyler: Is the MemoStyler for Python files syntax checking.

TAdvSQLMemoStyler: Is the MemoStyler for SQL files syntax checking.

TAdvWebMemoStyler: Is the MemoStyler for Web files syntax checking.

TAdvXMLMemoStyler: Is the MemoStyler for XML files syntax checking.

TAdvMultiFileMemo*: Component for directly handling the editing of multiple files at once (only available in TMS Component Pack, not in standalone TAdvMemo)

Online references

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

TMS TAdvMemo page:
http://www.tmssoftware.com/site/advmemo.asp

TMS manuals:
http://www.tmssoftware.com/site/manuals.asp

TAdvMemo

TAdvMemo description

The TMS TAdvMemo is a highly configurable syntax highlighting memo control with support for code-folding.

The TMS TAdvMemo has features like auto completion and auto correction, bookmarks support, markers and breakpoint support and holds a source code container.

The TMS TAdvMemo control supports a wide range of programming s.

The TMS TAdvMemo optimized for speed text rendering is based on the use of fixed-width TTF fonts. Make sure to specify a fixed-width TTF font for TAdvMemo.

TAdvMemo features

- Lightweight memo control with configurable syntax highlighting
- Highlighting for Basic, C#, CSS, Emotion, HTML, INI, JavaScript, Pascal, Perl, PHP, Python, SQL, Web and XML files.
- Codefolding support
- Undo and redo functions
- Optional gutter with configurable line number display
- Clipboard operations
- Find and replace dialogs
- Printing support
- Save to formatted HTML support
- DB-aware version included
- Configurable parameter hinting
- Configurable auto-completion
- URL aware
- Error marking
- Styler available for emoticons
- Office styles, Metro style and compatible with TMS TAdvFormStyler / TAdvAppStyler
- Code list component
- Separate memo contents container for easy switching of code with single memo
- Modified line indication
- Regular selection and block selection mode
- Search highlight, search with expressions
- Support for spell checking with Addict Spell Checker (See http://www.addictive-software.com)
- Also part of the TMS Component Pack

TAdvMemo architecture
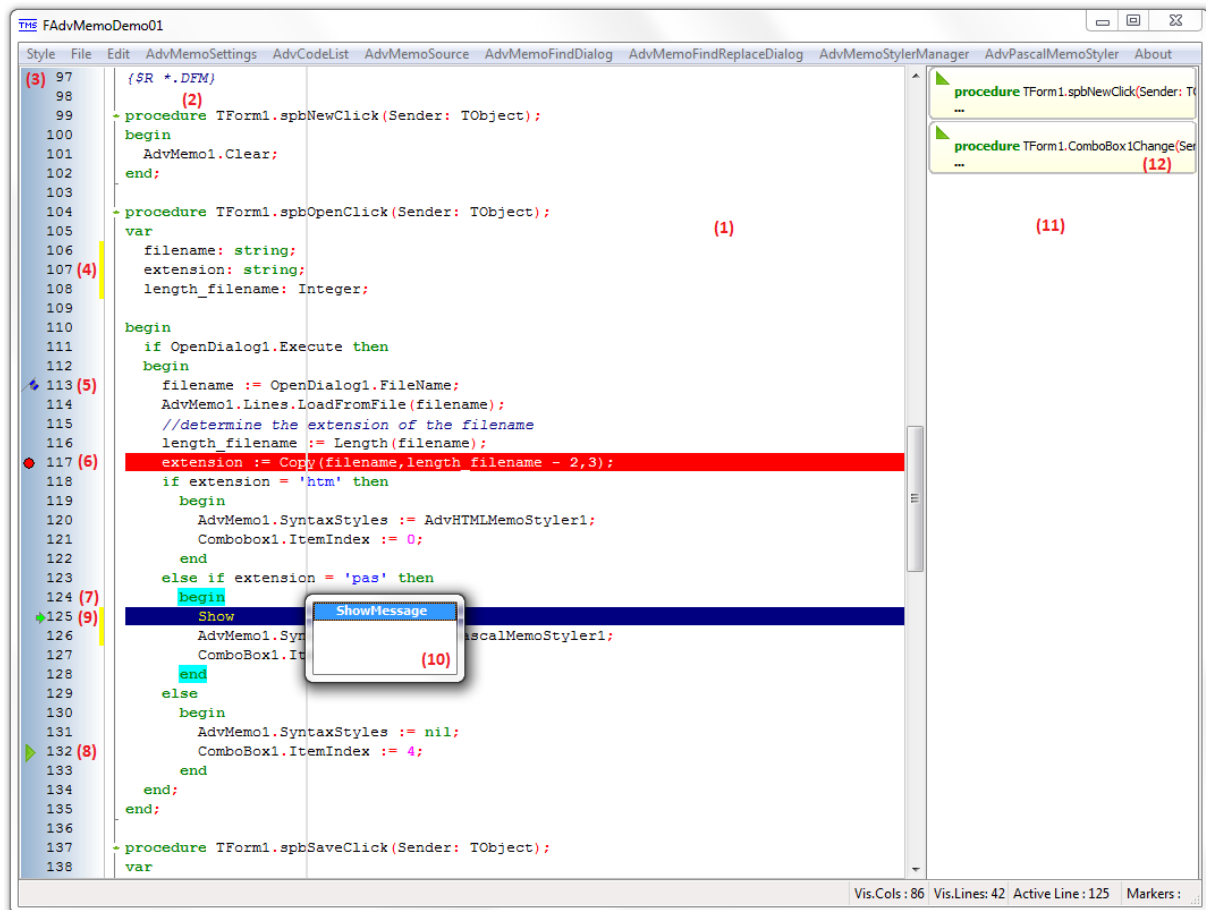
The core part of the TAdvMemo control is the memo field (1), note syntax highlighting (2). A gutter can be created (3), holding additional information like a modified text indicator (4), a bookmark (5), a breakpoint (6), code block indication (7) and a marker (8).

The active line (9), can be indicated with an active line indicator (green arrow on the left hand side) and/or a highlighted line.

Code completion (10) can be activated by pressing the Ctrl + Spacebar keys simultaneously can appear n optional

On the right hand side, an example of a TAdvCodeList control (11), holding Code block items (12). Code blocks can be dragged and dropped from and to the TAdvMemo control.
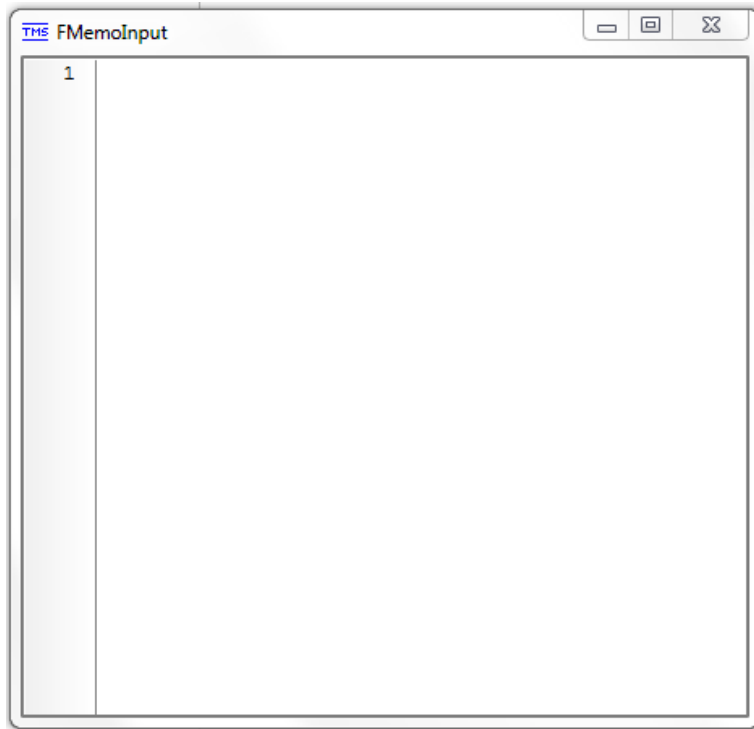
TAdvMemo use

Getting started

From the component palette, select the TAdvMemo control and drop it on a form.

This shows an empty memo, with a gutter on the left hand side.

and the selected text with the property AdvMemo.Selection: string. The selection can also be retrieved in X,Y coordinates with AdvMemo.SelStartX, AdvMemo.SelStartY, AdvMemo.SelEndX, AdvMemo.SelEndY.

General TAdvMemo settings

published TAdvMemo properties

- **AcceptFiles:** Boolean: When set to true, the TAdvMemo control accepts files being dragged from Windows Explorer on the TAdvMemo canvas.

- **ActiveLineSettings:** Is a collection of TAdvActiveLineSettings. (More detailed information can be found in the TActiveLineSettings paragraph).

- **AutoCompletion:** Is a collection of TAutoCompletion . (More detailed information can be found in the TAutoCompletion paragraph).

- **AutoCompletionListImages:** Can be linked to a standardhe specification of Image-list. Images from this list can be shown on the left hand side of the individual autocompletion items.

- **AutoCorrect:** Is a collection of TAutoCorrect (More detailed information can be found in the TAutoCorrect paragraph).

- **AutoExpand: Boolean:** When set to true, when clicked after the end of the line, the line will be filled with spaces until the cursor position.

- **AutoHintParameterDelay: Integer:** Sets the waiting time before the hint gets displayed (in milliseconds).

- **AutoHintParameterPosition: TAutoHintParameterPosition:** Sets the position of the hint parameter to one of these predefined valuesof either hpBelowCode or hpAboveCodeat cursor position, or below the line at cursor position.

- **AutoHintParameters: TAutoHintParameters:** Sets the hintparameters to the predefined values of

    - hpAuto .

    - , hpManual .

    - or hpNone ..

- **AutoIndent: Boolean:** When set to true, the next line will automatically receive the same indentation as the originating line.

- **AutoThemeAdapt: Boolean:** When set to true, the colors will automatically adapt to those of the Windows color settings.

- **BandColor: TColor:** Sets the color of the even horizontal bands.

- **BkColor: TColor:** Sets the background color of the TAdvMemo bands.

- **BlockColor: TColor:** Sets the color of the code block start and code block end.

- **BlockLineColor: TColor:** Sets the color of the blocks underline line.

- **BlockShow: Boolean:** When set to true, predefined code blocks beginning and end is displayed in BlockColor.

- **BreakPointColor: TColor:** Sets the background color of the breakpoint line.

- **BreakPointTextColor: TColor:** Sets the text color of the breakpoint lines.

- **CaseSensitive: Boolean:** When set to true, the TAdvMemo control is case sensitive with respect to keyword highlighting.

- **CharCase: TCharCase:** Selects the default case of text entered in the memo (similar to a TEdit or TMemo)

- **ClipBoardFormats: TMemoClipboardFormat:** The following predefined clipboard formats can be enabled or disabled:

    - **cfText:** Standard text format.

    - **cfRTF:** Rich text format.

    - **cfHTML:** HTML text format.

When multiple clipboard formats are selected, the TAdvMemo will put a copy of the text in the different formats on the clipboard.

- **CodeFolding:** TCodeFolding type. (More detailed information can be found in the TAutoCorrect paragraph).

- **DelErase: Boolean:** When set to true, the selected area is erased when typing, when set to false only the first character where typed is erased.

- **EnhancedHomeKey: Boolean:** When set to true, the cursor goes to the start / end of the line without taking the spaces before or after the line in account.

- **Gutter:** TAdvGutter type. (More detailed information can be found in the TAdvGutter paragraph).

- **HiddenCaret: Boolean:** When set to true, does not show the caret (vertical line) where the cursor is positioned.

- **HideSelection: Boolean:** When set to true, the selection is hidden when the focus leaves the memo control.

- **Lines:** TAdvMemoStrings type. (More detailed information can be found in the TAdvMemoStrings paragraph).

- **MarkerList:** TAdvMarkerList type. (More detailed information can be found in the TAdvMarkerList paragraph).

- **MemoChecker: Boolean:** When set, allows to run extra validation routines like a spell checker to perform automatic word check as words are typed. (Example: TAdvMemoCapitalChecker or component to bind to Addict spell checker).

- **MemoSource:** TAdvMemoSource type. Can be used to assign different texts to one TAdvMemo list, instead of creating a TAdvMemo list for each text that is used on the form.

- **OleDropSource:** When true, the AdvMemo serves as source for OLE drag & drop operations. This means the selected text can be dragged to other OLE drag & drop enabled applications (like Microsoft Word, Microsoft Outlook, …)

- **OleDropTarget: TOleDropTarget = set of (odtFile,odtText):** When odtFile is selected, TAdvMemo serves as an OLE drop target for file OLE drag & drop operations. This means a file can be dragged from Windows Explorer for example and TAdvMemo will try to open this file. When odtText is selected, text dragged from OLE drop sources like Microsoft Word, Outlook can be directly dragged into TAdvMemo and the text will be inserted at the caret indicating the drop position. Note that both odtFile and odtText can be simultaneously selected.

- **PrintOptions:** TPrintOptions type. (More detailed information can be found in the TPrintOptions paragraph).

- **RightMargin: Integer:** Defines the position of the right margin.

- **RightMarginColor: TColor:** Sets the color for the vertical margin line.

- **SelBKColor: TColor:** Sets the background color for selected text.

- **SelColor: TColor:** Sets the font color for the selected text.

- **SelectionMode: TSelectionMode:** Selects between regular text selection mode (smText) and block selection mode (smBlock).

- **ShowRightMargin: Boolean:** When set to true, a right vertical margin is shown.

- **SmartTabs: Boolean:** When set to true, the tabs are automatically set to a multiple of the TAdvMemo.TabSize. When set to false, the TAdvMemo.TabSize is added from the current position of the cursor.

- **SyntaxStyles: TAdvCustomMemoStyler:** Sets the styler to one of the included syntax stylers components for automatic syntax based highlighting. One can choose out of the following included syntax stylers: AdvBasicMemoStyler, AdvCSharpMemoStyler, AdvCSSMemoStyler, AdvEmoticonMemoStyler, AdvHTMLMemoStyler, AdvINIMemoStyler, AdvJSMemoStyler, AdvPascalMemoStyler, AdvPerlMemoStyler, AdvPHPMemoStyler, AdvPythonMemoStyler, AdvSQLMemoStyler, AdvWebMemoStyler, AdvXMLMemoStyler. Alternatively, a custom syntax styler can be built by descending from TAdvCustomMemoStyler.

- **TabSize: Integer:** Sets the number of tabulation characters when pressing the Tab button.

- **TrimTrailingSpaces: Boolean:** When set to true, trailing spaces are removed.

- **UILanguage: TUILanguage:** Class property holding all text values used throughout the memo for easy internationalization.

- **UndoLimit: Integer:** Holds the maximum number of Undo steps.

- **UndoLineByLine: Boolean:** When set to true, an Undo is performed one line at a time.

- **UrlAware: Boolean:** When set to true, an URL is recognized by the control, and displayed in a specific style.

- **UrlStyle: TCharStyle:** Defines the style for the drawing of the Url string.

    - **BkColor: TColor:** Sets the background color of the Url string.

    - **Style: TFontStyle:** Style has four properties that can be enabled or disabled: fsBold, fsItalic, fsUnderline and fsStrikeOut.

    - **TextColor: TColor:** Sets the font color of the Url string.

- **UseStyler: Boolean:** When set to true, the assigned syntax styler settings will be applied to the TAdvMemo control.

- **WantTab: Boolean:** When set to true, instead of jumping to the next control, tabulation functionality is attributed to the TAdvMemo control.

- **WordWrap: TWordWrapStyle**

- public **CurX: Integer:** Returns the column position on the activeline.

- **CurY: Integer:** Returns the row position of the activeline.

- **Selection: String:** Gets / sets the selected text.

- **SelStart: Integer:** Returns the start position of the selected text.

- **SelLength: Integer:** Returns the length of the selected text

- **SelStartX: Integer:** Returns the starting column of the selected text.

- **SelStartY: Integer:** Returns the starting row of the selected text.

- **SelEndX: Integer:** Returns the ending column of the selected text.

- **SelEndY: Integer:** Returns the ending row of the selected text.

- **ExpandNode[Index: Integer]: Boolean;** When set to true, the node[index] is expanded.

- **ActiveLine: Integer** Returns the active line number.

- **Bookmark[LineIndex: Integer]: Boolean:** When set to true, the line at LineIndex position holds a bookmark.

- **Bookmarks[Index: Integer]: Integer:** Holds the line index for that bookmark number Index.

- **BookmarkIndex[LineIndex: integer]: Integer:** Holds the index number for the bookmark at LineIndex.

- **BreakPoint[Index: Integer]: Boolean :** When set to true, the line at index position holds a breakpoint.

- **LineModified[Index: Integer]: Boolean :** When set to true, the line at index position has been changed.

- **LineModifiedInt[Index: Integer]: TLineModifiedState :** Holds one of three possible values lmUnmodified, lmModified (results in a yellow line) or lmSaved (results in a green line)**.**

- **Executable[Index: Integer]: Boolean;** When true, the line at position Index holds executable code.

- **OverWrite: Boolean:** When set to true, the entered text overwrites existing one, if set to false, text is inserted at the cursor position.

- **SelectSingleLine: Boolean**

- **TopLine: Integer**

- **LeftCol: Integer**

- **Modified: Boolean**

- **WrappedText: String**Returnstext of the memo as

- **WrappedLine[Index: Integer]: String**Returns a line Index within a memo with wording applied.

- **UndoList: TAdvUndoList**, holding all actions for undo in the memo.

- **Version: String**Returnsversion of the as string.

- **VisiblePosCount:** Returns the number of visible columns of the control.

- **VisibleLineCount:** Returns the number of visible lines of the control.

- **MarkerCount:** Returns the number of markers that were created.

-

TAdvMemo methods

- **Procedure RefreshMemo:** Reapplies syntax styling and repaints the memo.

-

- **Procedure MouseToCursor(X, Y: Integer; var CursorX, CursorY: Integer):** Returns the text-column CursorX and text-line CursorY position for the given X, Y mouse position in the editor.

- **Procedure UpdateWrap:** Reapplies wordwrapping in the memo.

- **Function GetWrappedLineIndex(Index: Integer): Integer;** Gets the wordwrapped line index of a line at position Index when no wrapping is applied.

- **Procedure CopyToClipboard:** Copies the selected text to the clipboard.

- **Procedure PasteFromClipboard:** Inserts the text from the clipboard to the TAdvMemo control.

- **Procedure CutToClipboard:** Cuts the selected text to the clipboard.

- **Function IsEmpty: Boolean:** Returns True if the TAdvMemo.Editor is empty.

- **Procedure SelectAll:** Selects al the text in the TAdvMemo editor.

- **Procedure DeleteLine:** Deletes the line at the current cursor position.

- **Procedure ClearBreakPoints:** Clears the breakpoint list, resulting in the removing of all breakpoints.

- **Procedure ClearModified:** Clears the modified-lines list, resulting in the removing of the modified line indicators in the gutter for all lines.

- **Procedure ClearExecutableLines:** Clears the modified-lines-list, resulting in the removing of the executable line indicators in the gutter.

- **Procedure SetError(LineNo, ErrorPos, ErrLen: Integer) :** Underlines the text on line LineNo from position ErrorPos for ErrLen characters.

- **Procedure ClearErrors:** Clears all marked errors in the editor.

- **Procedure ClearLineErrors (LineNo: Integer):** Clears all errors at line number LineNo.

- **Procedure ClearWordError (LineNo, LinePos: Integer):** Clears the error in the word at position LineNo and LinePos.

- **Function CharFromPos(X, Y: Integer): TFullPos:** Returns the exact position of the character at the actual mouse cursor position.

- **Procedure PosFromText(TextPos: Integer, var X, Y: integer): //**Converts the TextPos text position in the memo to X,Y cursor coordinates.

- **Procedure TextFromPos(X, Y: Integer; var TextPos: Integer):** Converts the cursor coordinates X,Y to the text position in the memo.

- **Procedure DeleteSelection:** current/selected text in

- **Procedure InsertText(AValue: String):**

- **Procedure InsertTextAtXY(AValue: String); X, Y: Integer);**

- **Procedure DeleteTextAtXY(X, Y, Numchar : Integer);**

- **Procedure BlockIndent(Fromline, ToLine, Indent: Integer; AllowUndo: Boolean = True)** FL in characters.

- **Procedure GetMarkers(Markers:: TAdvMarkers): Boolean:** Returns a collection of all markers in the memo.

- **Function MarkerAtLine(LineNo: Integer): Boolean**

- **Procedure SaveMemoSettingsToFile(FileName:String):** Saves the settings of the memo to a file.

- **Procedure LoadMemoSettingsFromFile(FileName:String):** Loads the settings of the memo from a file.

- **Function WordAtCursor: string:** Returns the word at the actual cursor position.

- **Function WordTillCursor: string:** Returns the word ending at the actual cursor position.

- **Function WordAtCursorPos(var Pos:Integer): string:** Returns the word at text position Pos and updates text position to the start position of the word in the memo.

- **Function WordAtXY(X, Y: Integer): string:** Returns the word at the given X,Y cursor coordinates in the memo.

- **Function TokenAtXY(X, Y: Integer): string:** Returns the token at the given X,Y cursor coordinates in the memo.

- **Function FullWordAtXY (X, Y: Integer): string:** Returns the full word at the given X,Y cursor coordinates in the memo.

- **Procedure ClearSelection:** Clears the current selection in the memo.

- **Procedure ClearBookmarks:** Clears the bookmark list, removing all previously added bookmarks.

- **Procedure ClearUndoRedo:** Clears the undo-redo list.

- **Procedure GotoBookmark(Index: Integer):** Positions the cursor at the chosen bookmark.

- **Function HasBookmarks: Boolean:** Returns true if bookmarks have been created for the TAdvMemo.

- **Function HasMarkers: Boolean:** Returns true if markers have been created for the TAdvMemo.

- **Function FindTextCount(SearchStr: String, Options: TFindOptions): Integer:** Returns the number of occurrences of SearchStr, depending on the parameters set in TFindOptions: frDown, frFindNext, frHideMatchCase, frHideWholeWord, frHideUpDown, frMatchCase, frDisableMatchCase, frDisableUpDown, frDisableWholeWord, frReplace, frReplaceAll, frWholeWord, frShowHelp.

- **Function FindText(SearchStr: String, Options: TFindOptions): Integer:** Returns the position of searchstr in the memo, depending on the parameters set in TFindOptions, as described in FindTextCount.

- **Overload Function FindText(SearchStr: String, Options: TFindOptionsEx): Integer:** Extended FindText function returns the number of occurrences of SearchStr, depending on the parameters set in TFindOptionsEx (has two extra options to perform search only in selected text and use a mask expression to search for text): freDown, freFindNext, freHideMatchCase, freHideWholeWord, freHideUpDown, freMatchCase, freDisableMatchCase, freDisableUpDown, freDisableWholeWord, freReplace, freReplaceAll, freWholeWord, freShowHelp, freSelection, freExpression.

- **Function FindTextInMemo (SearchStr: String, Options: TFindOptions): Integer:** Returns the position of SearchStr in the memo, depending on the parameters set in TFindOptions, as described in FindTextCount.

- **Function FindTextPos (SearchStr: String, Options: TFindOptions): Integer:** Returns the position of SearchStr in the memo, depending on the parameters set in TFindOptions, as described in FindTextCount.

- **Function FindAndReplace (SearchStr, NewStr: String, Options: TFindOptions): Integer:** Searches for SearchStr in the editor, depending on the parameters set in TFindOptions, and replaces the SearchStr with NewStr text.

- **Overload Function FindAndReplace (SearchStr, NewStr: String, Options: TFindOptionseEx): Integer:** Searches for SearchStr in the memo, depending on the parameters set in TFindOptionsEx, and replaces the SearchStr with NewStr text.

- **Procedure Clear:** Clears all text from the TAdvMemo editor.

- **Procedure Undo:** Reverses the last performed action in the memo. The number of undo-entries is limited with the TAdvMemo.UndoLimit property;

- **Procedure Redo:** Reapplies the last reversed action.

- **Function CanUndo: Boolean;**

- **Function CanRedo: Boolean;**

- **Function CanCut: Boolean;**

- **Function CanPaste: Boolean;**

- **Procedure GotoMarker(Marker: Integer)**

- **Procedure GotoMarkerName(MarkerText: String);**

- **Procedure AddMarker(LineIdx, ImageIdx: Integer)** for display in the gutter

- **Procedure AddMarker(LineIndex, ImageIndex: Integer; MarkerText: String)** for display in the gutterT

- **Procedure RemoveMarker(LineIndex: Integer);**

- **Procedure ClearAllMarkers** l

- **Function WordIsUrl(s: String): Boolean**

- **Function AddCodeFolding(StartLineIndex, EndLineIndex: Integer): Boolean** A node will appear at line StartLineIndex from where expand/collaps of a code block is possible.

- **Procedure RemoveCodeFolding(StartLineIndex: Integer)**

- **Procedure RemoveAllCodeFolding** (nodes)

- **Function IsNode(LineIndex: Integer): Boolean**

- **Procedure ExpandAllNodes**

- **Procedure CollapsAllNodes**

- **Procedure ToggleNode(LineIndex: Integer);**

- **Procedure AutoCodeFold;**definitionsusedsyntax

- **Function SaveToHTML(Filename: String; Fixedfonts: Boolean = True): Boolean**HTML

- **Function SaveToHTMLStream(AStream: TMemoryStream; Fixedfonts: Boolean = True): Boolean**HTML

- **Function SaveToRTF(Filename: String; Fixedfonts: Boolean = True): Boolean**

- **Function SaveToRTFStream(AStream: TMemoryStream; Fixedfonts: Boolean = True): Boolean**

- **Function CopyHTMLToClipboard**

- **Function NumberOfPages(ACanvas: TCanvas; PageWidth, PageHeight, PageNr: Integer): Integer;** Returns the nr. Of pages that would be needed to print the memo from Page PageNr on a page with size PageWidth, PageHeight.

- **Function PrintToCanvas(ACanvas: TCanvas; PageWidth, PageHeight, PageNr: Integer) Integer;** Renders the memo page PageNr on a TCanvas with size PageWidth, PageHeight.

- **Procedure PrintPages(FromPage, ToPage: Integer);**

- **Procedure Print;**

- **Procedure PrintSelection;**

- **Procedure BeginUpdate;**number.

- **Procedure EndUpdate;**

- **Function GetVersionNr: Integer;**

- **Function GetVersionString: String;**

- **Procedure SetStyle(AStyle: TAdvMemoStyle);**

- **Procedure SetComponentStyle(AStyle: TTMSStyle)**ITMSStyle interface method. Se**;**

- **Procedure SetColorTones(Atones: TColorTones):** ITMSTones interface method. Configures the memo in a Metro color set.

- the active line changedstartedstarted and allows for dynamic customization of this list needed, allows for dynamic customization of parameter hints.when a occurred a k occuredneeds to be and allows for custom drawing on the gutter.neededneeded ., allowing to dynamically alter the line background...

- **OnOleDropFile:** event is triggered when a file is dropped on the memo. Only single file drop is allowed. When multiple files are dropped on the memo, only the first file will be used. The filename of the file dropped is returned in the event and an Allow parameters enabled to accept or block the opening of the file dropped in TAdvMemo. It is required that odtFile is set in AdvMemo.OleDropTarget for this event to be triggered.

- **OnOleDropText:** event is triggered when text is dropped on the memo. The event returns the cursor coordinates where the text is dropped. The text dropped is returned via the Text parameter and the Allow parameter enables to allow or block the actual text drop. It is required that odtText is set in AdvMemo.OleDropTarget for this event to be triggered.

- **OnOleTextDragged:** event triggered when text is dragged from the memo to an OLE drag & drop enabled application. a through a find & replace actionvertical or horizontal scrolling in the memo is done to display a hint on the scrollbar with extra information during the scrolling.in the memo hamemo via a Find callTAdvActiveLineSettings

The TAdvActiveLineSettings class property controls how highlighting of the line position of the cursor is done. This highlighting can be done by a colored background and font, or by displaying an arrow in the gutter on the left. For maximum visibility, both possibilities can be set at the same time.

TAdvActiveLineSettings properties

- **ActiveLineAtCursor:** When set to true, the active line at the cursor position is highlighted.

- **ActiveLineColor:** Holds the background color for the active line.

- **ActiveLineTextColor:** Holds the font color for the active line.

- **ShowActiveLine:** When set to true, displays the active line in color, as defined with ActiveLineColor and ActiveLineTextColor.

- **ShowActiveLineIndicator:** When set to true, shows a green arrow next to the line number in the gutter.

TAutoCompletion

The TAutoCompletion collection property allows to control the automatic completion of the edited text, depending on the chosen TAdvMemo.SyntaxsStyles component. Auto completion can be triggered with the keyboard by pressing Ctrl-Space.

TAutoCompletion properties

- **Active:** When set to true, the auto completion feature is activated.

- **AutoDisplay:** When set to true, the auto completion listbox is shown on the screen after TAutoCompletion.Delay milliseconds.

- **AutoWidth:** When set to true, the width of the auto completion listbox is adapted according to the width of the items it holds.

- **Color:** Sets the color for the auto completion listbox.

- **ColorEvent:** Sets the font color for the 'Event' text in the listbox.

- **ColorFunc:** Sets the font color for the 'Function' text in the listbox.

- **ColorIdentifier:** Sets the font color for the 'Identifier' text in the listbox.

- **ColorMethod:** Sets the font color for the 'Method' text in the listbox.

- **ColorProc:** Sets the font color for the 'Procedure' text in the listbox.

- **ColorProp:** Sets the font color for the 'Property' text in the listbox.

- **ColorVar:** Sets the font color for the 'Var' text in the listbox.

- **Delay:** Sets the time delay before displaying the auto completion listbox.

- **Font:** Sets the font to be used in the auto completion listbox.

- **FromFirstChar:** When set to true, auto completion can start immediately from the first entered character.

- **Height:** Sets the height of the auto completion listbox.

- **KeepLastSize:** When set to true, displays the auto completion listbox at the same size as defined when last displayed.

- **MaxWidth:** Sets the maximum with of the auto completion listbox.

- **ShowImages:** When set to true, displays the predefined images at the left hand side of the listbox items. Images are set via the imagelist assigned to AdvMemo.AutoCompletionListImages.

- **SizeDropDown:** When set to true, the dropdown of the auto completion listbox can be sized by dragging the bottom right corner.

- **StartToken:** Holds the start characters that trigger the display of the auto completion listbox

- **Width:** Sets the width of the auto completion listbox.

TAutoCorrect

TAdvMemo.AutoCorrect class property controls the optional auto correction capabilities in the memo. This comprises the possibility of creating a list of incorrect words, and their correction. Click the TAdvMemo.AutoCorrect property and a collection editor will popup, allowing you to add new or remove existing items.
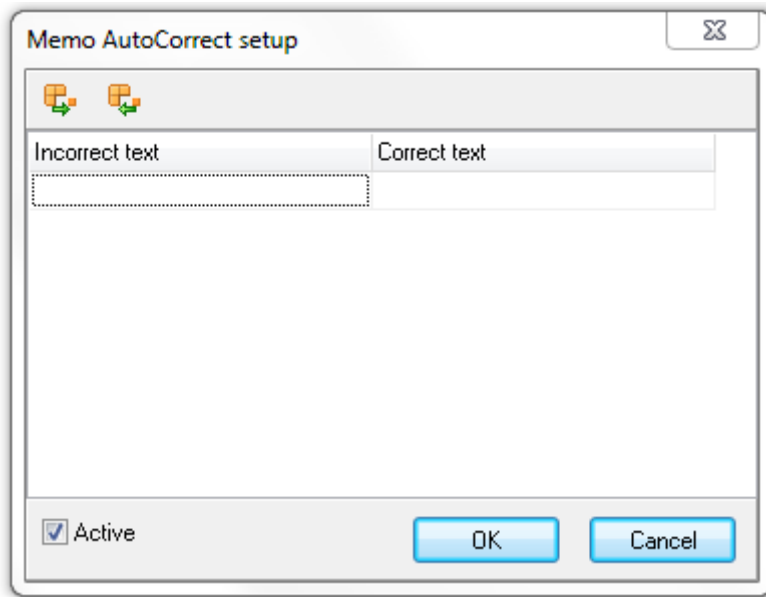
When Active is set to true, incorrect entries will be corrected according to the data in the entered in the Memo AutoCorrect setup.

TAutoCorrect methods

- **Procedure CheckLine(LineNo:Integer); :** Checks line with position LineNo for errors.

- **Procedure CheckWord(LineNo, LinePos: Integer; s:string); :** Checks the word at position LineNo – LinePos for errors. S holds the word at position.

- **Procedure CorrectLine(LineNo: Integer); :** Corrects errors in the line with position LineNo.

- **Procedure CorrectWord(LineNo, LinePos: Integer; var s:string); :** Corrects errors in the word at position LineNo. S holds the new value for the word.

- **Procedure CheckAllLines;** Checks all lines within the memo control.

- **Procedure CorrectAllLines;** Corrects all lines within the memo control.

TCodeFolding

The codefolding property makes it possible to group and expand or collaps this group of lines with a node.

TCodeFolding properties

- **CollapsedGlyph:** Sets the image displayed next to a folded code region when the folded code is in collapsed state.

- **Enabled:** When set to true, the code folding feature is activated.

- **ExpandGlyph:** Sets the image displayed next to a foldable code region when the folded code is in expanded state.

- **LineColor:** Sets the color of the code folding line along which the nodes are displayed.

TAdvGutter

The TAdvGutter is a collection of properties defining the layout of the gutter, a vertical pane on the left side of the memo control.

TAdvGutter properties

- **BorderColor:** Sets the color of the TAdvGutter border.

- **DigitCount:** Sets the number of leading zeros used in line numbers.

- **Font:** Sets the default font that will be used for the gutter.

- **GutterColor:** Sets the start gradient color of the gutter.

- **GutterColorTo:** Sets the end gradient color of the gutter.

- **GutterMargin:** Sets the left margin for the text in the gutter.

- **GutterWidth:** Sets the overall width of the gutter.

- **LineNumberAt:** Default this value is 1, meaning that the line number is shown on every line. When LineNumberAt would be set to 10, this would cause the line number to be displayed only every 1Oth line.

- **LineNumberStart:** Sets the start value of the line number. This allows to override the default line numbering that starts from line 1.

- **LineNumberTextColor:** Sets the font color of the gutter line numbers.

- **ModifiedColor:** Sets the color displayed next to modified lines.

- **NumberSuffix:** Defines a suffix for the numbers.

- **ShowLeadingZeros:** When set to true, line numbers are displayed with leading zeroes, as many as defined in the DigitCount.

- **ShowLineNumbers:** When set to true, line numbers are displayed in the gutter.

- **ShowModified:** When set to true, a vertical line is shown in ModifiedColor, for the lines of text that have been changed.

- **Visible:** When set to true, the gutter pane is displayed.

TAdvMemo.Lines

Adding or removing lines can be done either programmatically or at design-time. Double-click the TAdvMemo.Lines property and the TAdvMemo editor will popup, allowing adding new or removing existing lines.
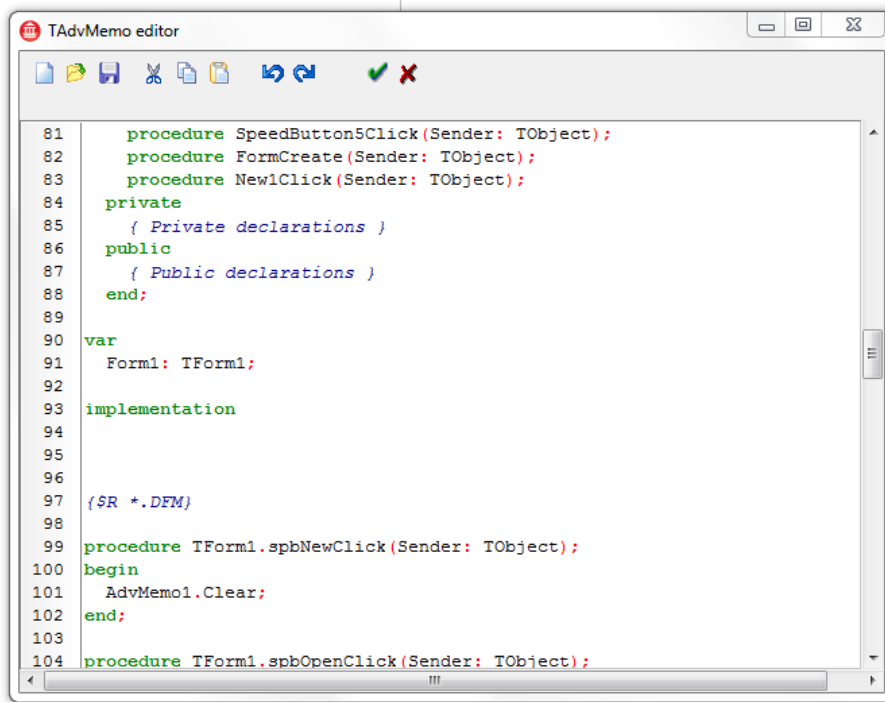
A text can be copy-pasted to the TAdvMemo Editor. Drag&drop of a file is not possible.

Adding/Removing lines programmatically

Adding a line:

```
AdvMemo1.Lines.Add('Manually added line');
```

Removing the active line:

```
AdvMemo1.Lines.Delete(AdvMemo1.ActiveLine);
```

Loading a file:

```
if OpenDialog1.Execute then
  AdvMemo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

TAdvMarkerList

TAdvMarkers can be used to display image markers in the gutter.

TAdvMarkerList properties

- **DefaultMakerImageIndex:** Holds the default marker image (by default -1 = no image).

- **ImageTransparentColor:** Holds the color that will be used for transparency.

- **MarkerImageList:** Holds the name of the image list that will be used to display the markers.

- **UseDefaultMarkerImageIndex:** When set to true, all markers use the same imageindex, i.e. DefaultMarkerImageIndex.

A marker can be added programmatically by using the following code:

```
AdvMemo.MarkerList.AddMarker(LineIdx, ImageIdx);
```

A marker is created at the LineIdx position, and the image with index ImageIdx is used from the MarkerImageList.

Exporting to PDF

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

Then simply call:

AdvMemoPDFIO.Save(FileName);

TAdvMemoPDFIO 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 TAdvMemo content.

TPrintOptions

The TPrintOptions class property holds the settings that control how the memo will be printed.

TPrintOptions properties

- **Jobname:** Holds the name of the print job in the Windows print manager.

30

- **MarginBottom:** Sets the bottom margin for the printout.

- **MarginLeft:** Sets the left margin for the printout.

- **MarginRight:** Sets the right margin for the printout.

- **MarginTop:** Sets the top margin for the printout.

- **PageNr:** When set to true, a page number is printed.

- **PagePrefix:** Holds the text that will be printed before the page number (i.e. 'Page : ').

- **PrintLineNumbers:** When set to true, the line numbers will be printed.

- **Tilte:** Holds the page title string

TAdvMemo events

- **OnActiveLineChange:** Is triggered when moving to another line.

- **OnAutoCompletion:** Is triggered when AutoCompletion is executed.

- **OnBeforeAutoCompletion:** Is triggered before AutoCompletion is executed.

- **OnCancelAutoCompletion:** Is triggered when AutoCompletion is canceled.

- **OnClipboardAction:** Is triggered on Ctrl + C is pressed, and selected text is moved to the clipboard.

- **OnColumnChange:** Is triggered when the cursor is moved left or right.

- **OnCursorChange:** Is triggered whenever the cursor position changes.

- **OnFileDrop:** Is triggered when a file is dropped on the canvas.

- **OnGetAutoCompletionList:** Is triggered when the autocompletion list is loaded.

- **OnGetAutoCompletionListIndex:** Is triggered when the autocompletion listindex is loaded.

- **OnGetParameterHint:** Is triggered when the parameterhint is loaded

- **OnGutterClick:** Is triggered when clicked in the gutter area.

- **OnGutterDblClick:** Is triggered when double clicked in the gutter area.

- **OnGutterDraw:** Is triggered when the gutter area is drawn.

- **OnGutterRightClick:** Is triggered when right clicked in the gutter area.

- **OnHintForToken:** Is triggered when a hint is found for the given token.

- **OnHintForWord:** Is triggered when a hint is found for the given word.

- **OnInsertAutoCompletionEntry:** Is triggered when an autocompletion entry is inserted

- **OnLineBkColor:** Is triggered when drawing the background color

- **OnMarkerAdded:** Is triggered when a marker is added

- **OnMarkerRemoved:** Is triggered when a marker is removed

- **OnOverWriteToggle:** Is triggered when the insert/overwrite is toggled.

- **OnReplace:** Is triggered when text is replaced.

- **OnScrollHint:** Is triggered when on scroll hint.

- **OnSelectionChange:** Is triggered when the selection is changed.

- **OnSortAutoCompletionList:** Is triggered on sorting the autocompletionlist

- **OnStartAutoCompletion:** Is triggered when starting (displaying) autocompletion.

- **OnStatusChange:** Is triggered when the status is changed

- **OnTextFound:** Is triggered when a searched for text is found.

- **OnUndoChange:** Is triggered when changes in the text are undone.

- **OnUrlClick:** Is triggered when an URL is clicked.

- **OnWordComplete:** Is triggered when a word is completed.

custom stylers

navigation options in memo

//?//

 Screenshot

An example on how to do this in program code can be found in the samples paragraph.

An example on how to do this in program code can be found in the samples paragraph.

TDBAdvMemo

The TMS TDBAdvMemo is the database aware version of the TMS TAdvMemo component and makes it possible to edit memo fields in databases. And this with all the extra coding features explained in the TAdvMemo paragraph.

TMS TDBAdvMemo has the same properties and events as the TMS TAdvDemo component, except for two properties to make the component database-aware.

TDBAdvMemo properties

- **DataField:** Holds the name of the datafield to bind to the memo.

- **DataSource:** Holds the name of the datasource where the datafield can be found.

TAdvCodeList

The TAdvCodeList is a list of code snippets that are shown with syntax highlighting in a listbox control. The TAdvCodeList can be used to hold code snippets and drag & drop these between the TAdvMemo and the TAdvCodeList. It can also be used as a clipboard monitor, displaying all code snippets that are being cut or copied to the clipboard.

General TAdvCodeList settings

TAdvCodeList properties

- **ClipboardAppend:** When set to true, new code blocks will be added at the end of the list, otherwise, new blocks will be inserted in top position.

- **ClipboardView:** When set to true, the TAdvCodeList will monitor all text that is cut or copied to the clipboard and display this in the code list.//

- **CodeBlockCaptionColor:** Sets the color of the code block item caption.

- **CodeBlockCaptionTextColor:** Sets the color of the code block item caption text.

- **CodeBlockColor:** Holds the start gradient color of the code block item.

- **CodeBlockColorTo:** Sets the end gradient color of the code block item.

- **CodeBlocks:** Is a collection of TCodeBlocks (detailed information can be found in the CodeBlocks paragraph).

- **CodeBlockSelectedColor:** Sets the start gradient color of the selected code block item.

- **CodeBlockSelectedColorTo:** Sets the end gradient color of the selected code block item.

- **CodeBorderColor:** Sets the color of the code block border item.

- **CodeBorderSelectColor:** Sets the color of the selected code block item border.

- **CodeBorderWidth:** Sets the width of the code block item border.

- **CodeIdent:** Sets the indentation value for the code block items (in characters).

- **Extendedselect:** When set to true, also selects beginning and trailing spaces.

- **IntegralHeight:** When set to true, the height of the listbox control will automatically adapt to show an integral number of code blocks.

- **ItemHeight:** Sets the height of the code block item.

- **MultiSelect:** When set to true, allows selecting multiple code blocks simultaneously.

- **SelectionColor:** Sets the color of the selected code block items.

- **ShowSelection:** When set to true, the selection of codeblocs is displayed.

- **Sorted:** When set to true, the code block items are sorted alphabetically before being displayed.

- **SyntaxStyles:** Sets the style to one of the syntax styler components for display. One can choose out of the following possibilities: AdvBasicMemoStyler, AdvCSharpMemoStyler, AdvCSSMemoStyler, AdvEmoticonMemoStyler, AdvHTMLMemoStyler, AdvINIMemoStyler, AdvJSMemoStyler, AdvPascalMemoStyler, AdvPerlMemoStyler, AdvPHPMemoStyler, AdvPythonMemoStyler,. AdvSQLMemoStyler, AdvWebMemoStyler, AdvXMLMemoStyler.

TAdvCodeList events

- **OnBlockClick:** Is triggered when a code block is clicked.

- **OnBlockDoubleClick:** Is triggered when a code block is double clicked.

- **OnBlockDelete:** Is triggered when a code block is deleted.

- **OnBlockDropped:** Is triggered when a code block is dropped on the list.

- **OnBlockInsertFromClipboard:** Is triggered when a code block is inserted from the clipboard.

- **OnBlockRightClick:** Is triggered when a code block is right clicked.

- **OnMeasureItem:** Is triggered when a code block is drawn, and allows modifying the height of each item.

- **OnBlockDelete:** Is triggered when a code block is deleted.

TCodeBlocks settings

instances.TCodeblocks is a collection of Codeblock items.

Codeblock items can be added at design time or programmatically.

Adding/Removing Tcodeblock values

First open the codeblock collection editor by clicking the TAdvCodeList.CodeBlocks property in the Object Inspector. From here, codeblocks can be added or removed.
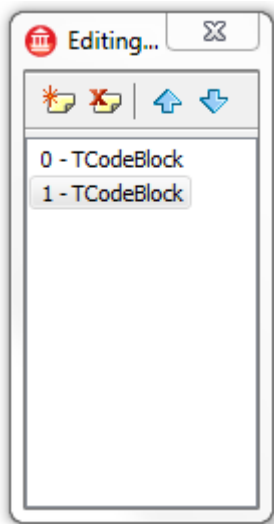
The equivalent in code is:

Adding a codeblock:

```
AdvCodeList1.Codeblocks.Add('Some code for the code block here');
```

or

```
var
  Block: TCodeBlock;
begin
  Block := AdvCodeList1.CodeBlocks.Add('');
  Block.Caption := 'Title';
  Block.Code.Text := 'Some code can be added here';
end;
```

Removing a codeblock:

```
AdvCodeList1.Codeblocks.Delete(Index);
```

TCodeBlock properties

- **Caption:** Holds the caption of the CodeBlock.

- **Code:** Is a TStringList holding the actual code for that block. Double clicking on the property opens the stringlist editor for adding and removing code text.
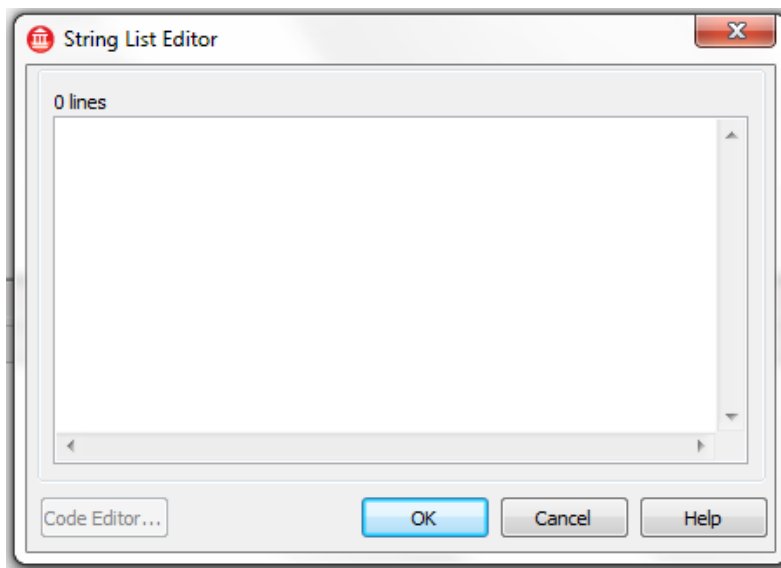
- **ImageIndex:** Sets the index value of the images from a TImagelist that can be optionally displayed in the caption of the codeblock.

TAdvMemoSource

The TDAdvMemo is the database aware version of the TMS TAdvMemo component and makes it possible to edit memo fields in databases. And this with all the extra coding features explained in the TAdvMemo paragraph.

The TMS TDBAdvMemo is a highly configurable syntax highlighting memo control with integrated code-folding.

The TMS TDBAdvMemo has features like auto completion, bookmarks support, and holds a source code container.

General TAdvMemoSource settings

TAdvMemoSource properties

- **Lines:** Is a TAdvMemoStrings holding the actual text for that TAdvMemoSource. Double clicking on the property opens the stringlist editor for adding and removing text lines.
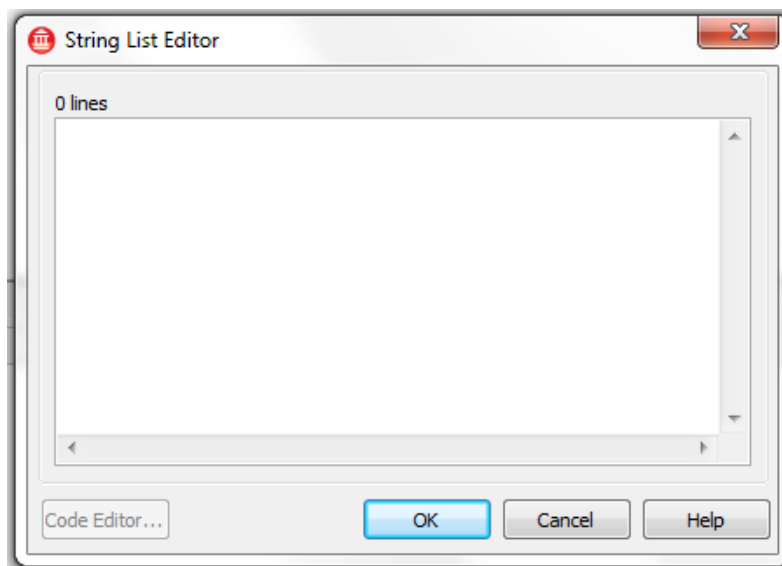
- **ReadOnly:** When set to true, the lines of the TAdvMemoSource cannot be edited when the MemoSource is used for TAdvMemo.

- **SyntaxStyler:** Sets the style to one of the predefined syntax highlighter components. One can choose out of the following possibilities: AdvBasicMemoStyler, AdvCSharpMemoStyler, AdvCSSMemoStyler, AdvEmoticonMemoStyler, AdvHTMLMemoStyler, AdvINIMemoStyler,

AdvJSMemoStyler, AdvPascalMemoStyler, AdvPerlMemoStyler, AdvPHPMemoStyler, AdvPythonMemoStyler, AdvSQLMemoStyler, AdvWebMemoStyler, AdvXMLMemoStyler.

This code demonstrates how 2 files can be loaded in a TAdvMemoSource instance and then assigned to a single TAdvMemo to edit each of the files:

```
begin
  AdvMemoSource1.Lines.LoadFromFile('c:\files\test.pas');
  AdvMemoSource1.SyntaxStyler := AdvPascalMemoStyler1;
  AdvMemoSource2.Lines.LoadFromFile('c:\files\test.cs');
  AdvMemoSource2.SyntaxStyler := AdvCSharpMemoStyler1;
  AdvMemo1.MemoSource := AdvMemoSource1;
end;
```

TAdvMemoFindDialog

The TAdvMemoFindDialog is a control that gives the possibility to search for a (sub)-string in the TAdvMemo. It can optionally bind to a more advanced find dialog TAdvFindDialog that also comes with TAdvMemo.

TAdvMemoFindDialog properties

- **AdvMemo:** The TAdvMemo control where the string will be searched for.

- **AutoHighlight**: When true, the matching occurences of strings will be automatically highlighted while typing in the TAdvFindDialog.

- **DisplayMessage:** When set to true, displays messages in the dialog box.

- **FindDialog:** Optionally can be set to an instance of TAdvFindDialog that offers extra functionality compared to the regular VCL TFindDialog.

- **FindText:** Holds the substring to find in the text.

- **FocusMemo:** When set to true, the focus is automatically given back to the memo when a string is found.

- **NotFoundMessage:** Holds the text that is displayed when the substring is not found.

- **Options:** Sets the various options to control the find action.

TAdvMemoFindDialog Methods

-   **Procedure Execute:** Starts the search in the TAdvMemo control for the (sub-) string defined in TAdvMemo.FindText.

-   **Procedure CloseDialog:** Close the CloseDialog programmatically.

TAdvMemoFindDialog Events

-   **OnFindDone:** Is triggered when the search is finished.

-   **OnFindText:** Is triggered when the text is found.

-   **OnShow:** Is triggered when the dialog is first shown.

-   **OnClose:** Is triggered when the dialog is closed.

TAdvMemoFindReplaceDialog

The TAdvMemoFindReplaceDialog gives the possibility to search for a certain string in the TAdvMemo-field, and to replace that found string with another one. It can optionally bind to a more advanced find dialog TAdvReplaceDialog that also comes with TAdvMemo.

TAdvMemoFindReplaceDialog properties

- **AdvMemo:** The TAdvMemo control where the string will be searched for.

- **AutoHighlight**: When true, the matching occurences of strings will be automatically highlighted while typing in the TAdvReplaceDialog.

- **DisplayMessage:** When set to true, displays messages in the dialog box.

- **FindText:** Holds the substring to find in the text.

- **FocusMemo:** When set to true, the focus is given to the memo.

- **NotFoundMessage:** Holds the text that is displayed when the substring is not found.

- **ReplaceDialog:** Optionally can be set to an instance of TAdvReplaceDialog that offers extra functionality compared to the regular VCL TReplaceDialog.

- **ReplaceText:** Holds the text used for replacing the FindText substring.

- **OnFindDone:** Is triggered when the search is finished.

- **OnFindText:** Is triggered when the text is found.

- **OnShow:** Is triggered when the dialog is first shown.

- **OnClose:** Is triggered when the dialog is closed.

TAdvMemoCapitalChecker

TAdvMemoCapitalChecker description

The TAdvMemoCapitalChecker is a component that will perform autocorrection to ensure that starting capital characters are used at the beginning of a sentence.

General TAdvMemoCapitalChecker settingsuse

Drop an instance of TAdvMemoCapitalChecker on the form and assign it to AdvMemo.MemoChecker. Once assigned, it will automatically perform the auto correction while typing in the TAdvMemo.

TAdvMemoStylerManager

The TAdvMemoStylerManager is a container for the TAdvMemoStylers.

General TAdvMemoStylerManager settings

TAdvMemoStylerManager properties

- **IncludeAllFiles:** When set to true, all files are included in the filter.

- **IncludeTextFiles:** When set to true, all text files will be included in the filter.

- **Items:** Is a collection of **TAdvMemoStylersCollectionItems** (detailed information can be found in the **TAdvMemoStylersCollectionItems** paragraph).

TAdvMemoStylerManager Methods

- **Function GetStylerByFileName (FileName: string): TAdvCustomMemoStyler:** Returns the syntax styler component associated with a filename.

- **Function GetStylerByName (strName: string): TAdvCustomMemoStyler:** Returns the syntax styler component based on the name of the component.

- **Function GetStyler (Idx: integer): TAdvCustomMemoStyler**: Returns the syntax styler component from the list created with GetStylerNames.

- **Function GetFilter (Idx: integer): string;** Returns the filter for selection of a file type associated with a syntax styler in an OpenDialog.

- **Function GetStylerNames (var List: TStrings): integer:** Returns the syntax styler component names in a list.

- **Function GetStylerItem (Idx: integer): TCollectionItem:** Returns the styler item at position idx.

- **Function GetStylerItemByFileName (FileName: string): TCollectionItem;** Returns the syntax styler component item of the collection associated with a filename.

- **Function GetStylerItemByName (strName: string): TCollectionItem;** Returns the syntax styler component item of the collection based on the name.

- **Function GetStylerStyleIndexByInfo (styler: TAdvCustomMemoStyler; Info: string): integer;** Returns the index of the syntax style component item in the collection basedon the info setting of the styler. //

- **Procedure SaveAllStylersToFile (FileName: string):** Saves all syntax styler component references to the given file name.

- **Procedure LoadAllStylersFromFile (FileName: string):** Loads all syntax styler component references from the given file name.

- **Procedure SaveStylerToFile (Styler: TAdvCustomMemoStyler; FileName: string):** Saves one syntax styler component reference to the given file name.

- **Procedure LoadStylerFromFile (Styler: TAdvCustomMemoStyler; FileName: string):** Loads one syntax styler component reference from the given file name.

- **Property ItemsCount:** Returns the number of items in the syntax styler component item collection.

TAdvMemoStylersCollection settings

TAdvMemoStylersCollection is a collection of TAdvMemoStylersCollectionItems.

TAdvMemoStylersCollectionItems can be added at design time or programmatically.

Adding/Removing TAdvMemoStylers

First open the TAdvMemoStylersCollection editor by clicking the TAdvMemoStylersCollection property in the Object Inspector. From here, TAdvMemoStylersCollectionItems can be added or removed.
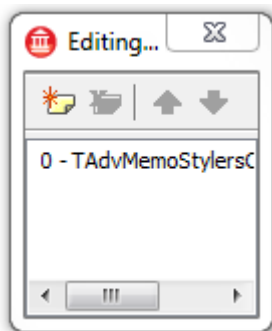
TAdvMemoStylersCollectionItem properties

- **FileName:** Holds the file filter to be associated with a syntax styler component.

- **Styler:** Gives the choice out of the existing stylers : AdvBasicMemoStyler, AdvCSharpMemoStyler, AdvCSSMemoStyler, AdvEmoticonMemoStyler,

AdvHTMLMemoStyler, AdvINIMemoStyler, AdvJSMemoStyler, AdvPascalMemoStyler, AdvPerlMemoStyler, AdvPHPMemoStyler, AdvPythonMemoStyler,. AdvSQLMemoStyler, AdvWebMemoStyler, AdvXMLMemoStyler.
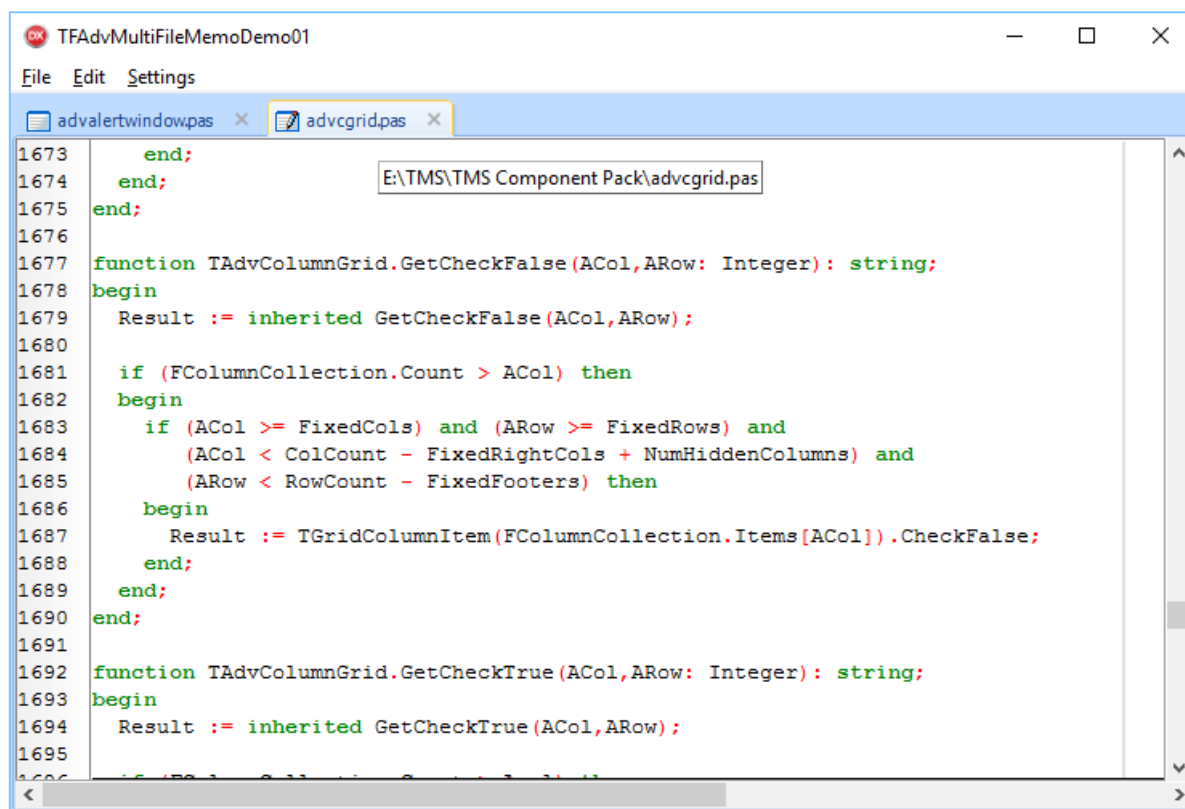
TAdvMultiFileMemo*

TAdvMultiFileMemo is a component that manages editing of multiple files simultaneously and presents these multiple files via a page control. TAdvMultiFileMemo can internally automatically manage applying the correct syntax styler with the opened files (based on filename extension). This means that when different file types are opened, different syntax stylers will be associated with the different files.

* TAdvMultiFileMemo is only available in combination with TMS Component Pack (http://www.tmssoftware.com/site/tmspack.asp)

121

TAdvMultiFileMemo features

- Easy to use methods for opening & closing files with automatic opening in new memo tab
- Automatic detection of syntax highlight rules based on file extension
- Optional automatic indication of modified tabs
- Optional automatic handling of file saving
- Optional automatic display of absolute file paths on tab hints

TAdvMultiFileMemo methods

TAdvMultiFileMemo.OpenMemoFile(Filename:string);

This adds a new tab and opens the specified filename in the memo in the new tab.

TAdvMultiFileMemo.OpenUntitledMemo(ACaption: string);

This adds a new tab with a new empty memo. Default the tab for this memo is set to 'Untitled' but the tab text can be customized via the ACaption parameter

TAdvMultiFileMemo.SaveAll;

Performs a save on all open & modified memos in the pagecontrol

TAdvMultiFileMemo.GetPageWithFile(Filename: string): TAdvMultiFileMemoPage;

Returns a reference to the TAdvMultiFileMemoPage that has the specified filename open

TAdvMultiFileMemo.GetModifiedPage: TAdvMultiFileMemoPage;

Returns the first page instance that has a modified memo

TAdvMultiFileMemo properties

TAdvMultiFileMemo.ActiveMemoPage: TAdvMultiFileMemoPage

Gets a reference to the active page in TAdvMultiFileMemo

TAdvMultiFileMemo.ActiveMemo: TAdvMemo

Gets a reference to the TAdvMemo instance on the active page in TAdvMultiFileMemo

TAdvMultiFileMemo.IsModified: Boolean;

Returns true if any of the memo instances is in modified state

TAdvMultiFileMemo.StylersBuiltin: TAdvMultiFileStylers;

Provides access to enable or disable the default built-in stylers Basic, C#, Pascal, HTML, Javascript, SQL. Via this set property it can be defined what built-in stylers can be automatically associated with opened file types.

TAdvMultiFileMemo.StylersCustom: TAdvMemoStylerManager;

Allows to associate a styler manager to the TAdvMultiFileMemo that is used to control syntax styler selection for the files that can be opened.

TAdvMultiFileMemo.DefaultStyler: TAdvCustomMemoStyler;

Allows to specify what syntax styler will be used when a new untitled memo is opened (and thus the file type is not yet known)

TAdvMultiFileMemo.ShowModified: Boolean;

When true, it will be indicated via a glyph in the page tab that a memo is in normal state or in modified state.

TAdvMultiFileMemo events

TAdvMultiFileMemo.OnInitMemo: TAdvMultiFileInitMemoEvent;

The event is triggered when a new TAdvMemo instance is created for opening a new file. A reference to the instance of the TAdvMemo is passed as a parameter and this can be used to customize the memo after opening.

TAdvMultiFileMemo.OnCloseMemoPage: TAdvMultiFileCloseMemoPageEvent;

This event is triggered when a user wants to close a page by clicking the close button the page tab. It returns the instance of the page and via the var parameter AllowClose it can be controlled whether the page will effectively close or not.

TAdvMultiFileMemo.OnSaveUntitled: TAdvMultiFileSaveUntitledEvent;

This method is triggered when a save is performed on an untitled memo page. This event passes a reference to the TAdvMultiFileMemoPage for which the save will be performed and a var parameter via which the filename is set.

TAdvMultiFileMemoPage

Each page in the TAdvMultiFileMemo control is of the type TAdvMultiFileMemoPage. The pages can be accessed via TAdvMultiFileMemo.AdvPages[Index: integer]: TAdvMultiFileMemoPage.

The TAdvMultiFileMemoPage has following properties:

property Filename: string : the name of the file opened in the page

property Memo: TAdvMemo : the instance of the memo in the page

property IsModified: Boolean : returns the modified state of the memo in the page

property IsUntitled: Boolean : returns true when the page contains an untitled memo

In addition, it also provides methods for opening or saving files:

function OpenFile(AFileName: string): boolean;

Opens file AFileName in the memo of the page

function OpenUntitled(ACaption: string): boolean;

Creates a new empty memo for a untitled file in the page

function HasFileOpened(AFileName: string): boolean;

Returns true when the memo has a file open

function SaveMemo(OnlyIfModified: boolean=false): boolean;

Saves the memo contents of this page

procedure SaveMemoAs(AFileName: string);

Saves the memo contents of this page to file with name AFileName.

Syntax Stylers

TAdvPascalMemoStyler description

The TAdvPascalMemoStyler is the syntax styler for Pascal files.

General TAdvPascalMemoStyler settings

- **TAdvPascalMemoStyler properties**
-

- **AllStyles:** Is a collection of TElementStyles (detailed information can be found in the TElementStyles paragraph).

- **AutoCompletion:** Is a TStringlist holding autocompletion text. When double clicking on the property, a string list editor is loaded, where autocompletionstrings can be added or removed.
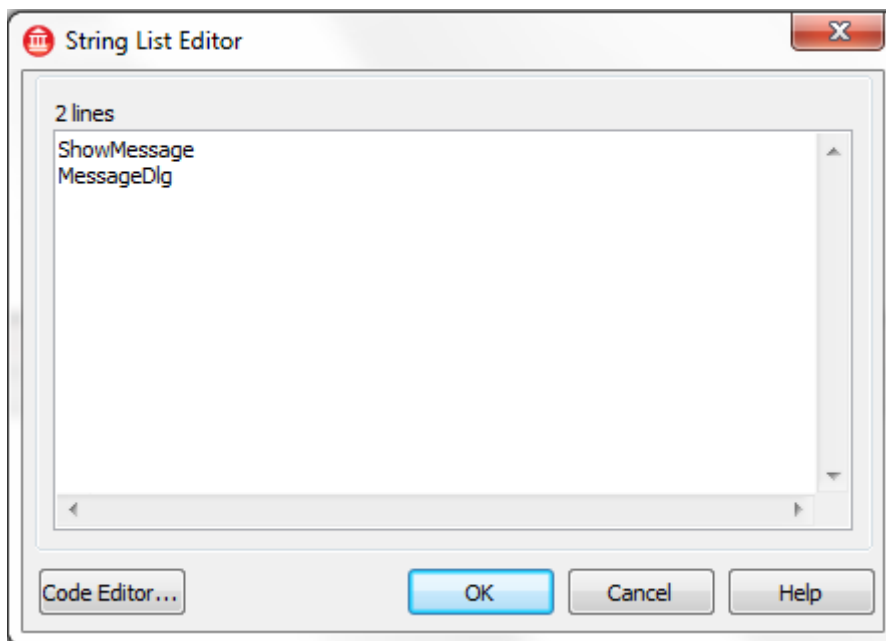
- **BlockEnd:** Holds the text that defines the end of a code block.

- **BlockStart** Holds the text(s, comma separated) that are recognized as the start of a code block.

- **CommentStyle:** Is of type TCharStyle, consisting of:

  - **BKColor:** Sets the background color

  - **Style:** Style has four properties that can be enabled or disabled: fsBold, fsItalic, fsUnderline and fsStrikeOut.

- **TextColor:** Sets the font color

- **DefaultExtension:** Holds the default extension for a Pascal File.

- **Description:** Holds a default description for the TAdvMemoStyler.

- **Extensions:** Holds the possible (point-comma delimited) valid Pascal file extensions.

- **Filter:** Holds a filter for use with an openfiledialog, containing the valid pascal file extensions.

- **HexIdentifier:** Holds the character that identifies a hex value.

- **HintParameter:** Is of type THintParameter, consisting of:

    - **BKColor:** Sets the background color.

    - **HintCharDelimiter:** Holds the hint character delimiter.

    - **HintCharEnd:** Holds the hint end character.

    - **HintCharStart:** Holds the hint start character.

    - **HintCharWriteDelimiter:** Holds the hint character write delimiter.

    - **Parameters:** Is a TStringlist, holding the values of the hint parameters.
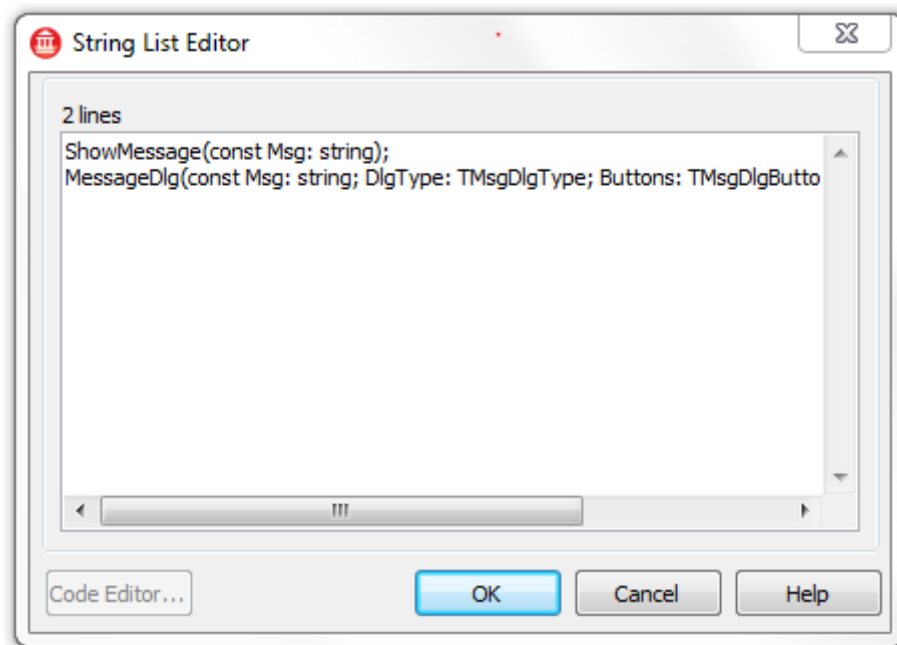
    - **TextColor:** Sets the font color for the hint.

    - **LineComment:** Holds the characters that identify the single line comment.

    - **MultiCommentLeft:** Holds the start character(s) that identifies the start of a multiline comment.

- **MultiCommentRight:** Holds the end character(s) that identify the end of a multiline comment.

- **Name:** Holds the name of the control.

- **NumberStyle:** Is of type TCharStyle, consisting of:

    - **BKColor:** Sets the background color

    - **Style:** Style has four properties that can be enabled or disabled: fsBold, fsItalic, fsUnderline and fsStrikeOut.

    - **TextColor:** Sets the font color.

- **RegionDefinitions:** Is a collection of TRegionDefinition (detailed information can be found in the TRegionDefinition paragraph).

- **StylerName:** Holds the name of the styler.

- **Tag:** Is an integer property that has no predefined meaning. It can be used for storing an additional integer value, or it can be typecast to any value such as a component reference or a pointer.

- **Version:** Holds the component version number.

TAdvPascalStyler methods

- **Procedure LoadFromDelphiSettings:** Loads typical Delphi color code settings from the windows registry.

TElementStyles settings

When double clicking on the property, a string list editor is loaded, where TElementStyles can be added or removed:
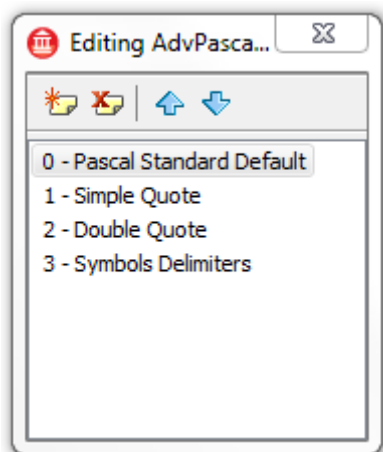
Different TElementStyles can be added to the collection. In the example of the TAdvPascalMemoStyler, following TElementStyles are needed:

0 – Pascal Standard Default: The keywords property holds all Delphi keywords.

1 – Simple Quote: Defines the simple quote character.

2 – Double Quote: Defines the double quote character.

3 – Symbols Delimiters: Defines the different symbols delimiters.

- **BracketEnd:** Holds the bracket end character, used when StyleType is stBracket. Note that when BracketEnd = #0, the end of the bracket style is considered when a word delimiter is found, i.e. a space or symbol.

- **BracketStart:** Holds the bracket start character, used when StyleType is stBracket.

- TElementStyles.Properties: Pascal Standard Default:

-

- **BGColor:** Sets the background color used for the syntax element.

- **CommentLeft:** identifier string for start of a multiline comment

- **CommentRight**: identifier string for end of a multiline comment

- **Font:** Sets the font used for the syntax element.

- **Info:** Holds the name of the TElementStyle.

- **KeyWords:** Is a TStringlist holding all keywords. When double clicking on the property, a string list editor is loaded where keywords can be added or removed. This is only used when the the StyleType of the TElementStyle is stKeyword.
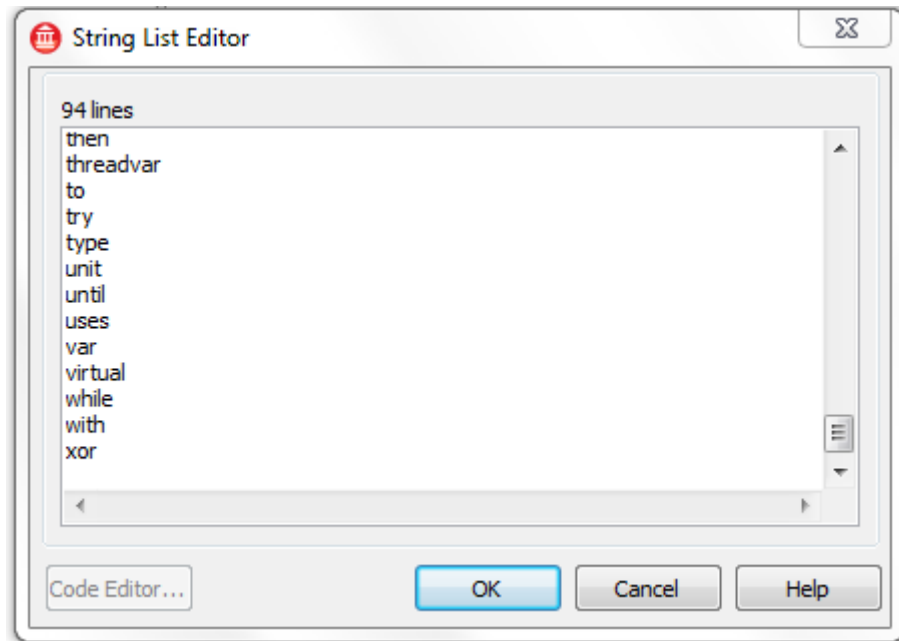
- **Symbols:** Holds the symbols when the StyleType is set to stSymbol.

- **StyleType:** Sets the StyleType to one of these values:

  o **stKeyWord:** keywords can be entered.

  o **stBracket:** bracket descriptions can be entered.

  o **stSymbol:** symbols can be entered.

  o **stComment:** comment left / right can be entered

TRegionDefinition settings

A region defines the start (and end) word of a foldable region when codefolding is activated.

- TRegionDefinition.Properties

- **Identifier:** Identifies the name of the keyword (i.e. Function).

- **RegionEnd:** Holds the last word (i.e. 'end').

- **RegionStart:** Holds the first word (i.e. 'begin').

- **RegionType:** Holds the type of the region and can be one of these values:

  o **rtClosed:** The region is ended by a RegionEnd statement.

  o **rtOpen:** The region has no RegionEnd statement.

  o **rtFile**: The region spans the entire file.//

- o **rtIgnore**: The RegionEnd statement is ignored.

- **ShowComments:** When set to true, comments are displayed.

TAdvBasicMemoStyler

The TAdvBasicMemoStyler is the syntax styler for MS Visual Basic files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvCSharpMemoStyler

The TAdvCSharpMemoStyler is the syntax styler for C-Sharp files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvCSSMemoStyler

The TAdvCSSMemoStyler is the syntax styler for CSS files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvEmoticonMemoStyler

The TAdvEmoticonMemoStyler is the syntax styler for text files that will display emoticon images instead of typical emoticon character sequences such as ;), :) , …

The properties are similar to the TAdvPascalMemoStyler.

-

TAdvHTMLMemoStyler

The TAdvHTMLMemoStyler is the syntax MemoStyler for HTML files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvINIMemoStyler

The TAdvINIMemoStyler is the syntax styler for INI files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvJSMemoStyler

The TAdvJSMemoStyler is the syntax styler for JavaScript files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvJSMemoStyler properties

TAdvPerlMemoStyler

The TAdvPerlMemoStyler is the syntax styler for Perl files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvPHPMemoStyler

The TAdvPHPMemoStyler is the syntax styler for PHP files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvPythonMemoStyler

The TAdvPythonMemoStyler is the syntax styler for Python files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvSQLMemoStyler

The TAdvSQLMemoStyler is the syntax styler for SQL files.
The properties are similar to the TAdvPascalMemoStyler.

TAdvWebMemoStyler

The TAdvWebMemoStyler is the syntax styler for Web files (mix of HTML ,CSS, JS).
The properties are similar to the TAdvPascalMemoStyler.

TAdvXMLMemoStyler

The TAdvXMLMemoStyler is the syntax styler for XML files.
The properties are similar to the TAdvPascalMemoStyler.

Demo 1: Configuration demo

The program shows the various configuration possibilities of the TAdvMemo component. It allows to interactively set various properties and the changes will be immediately reflected in the memo field.
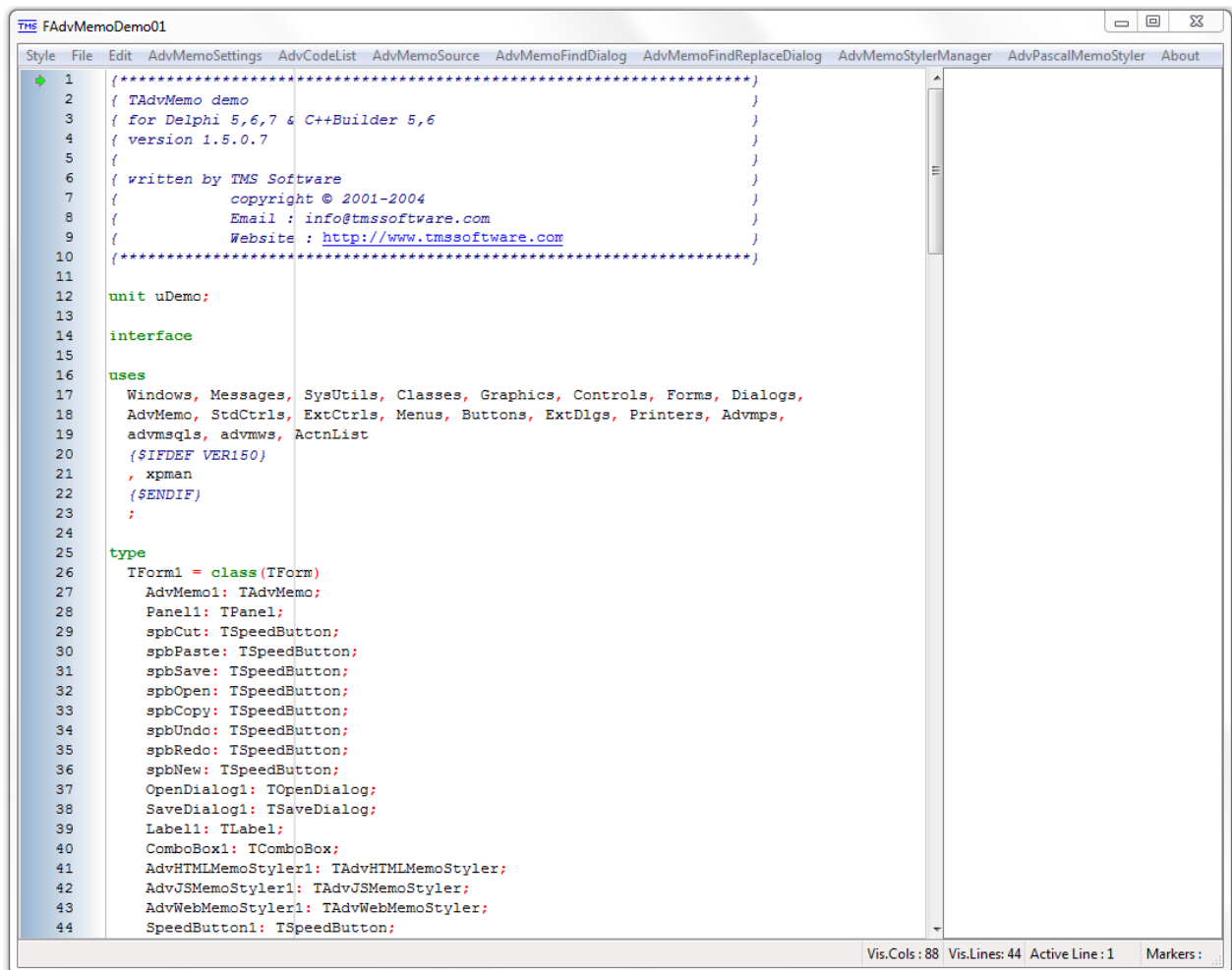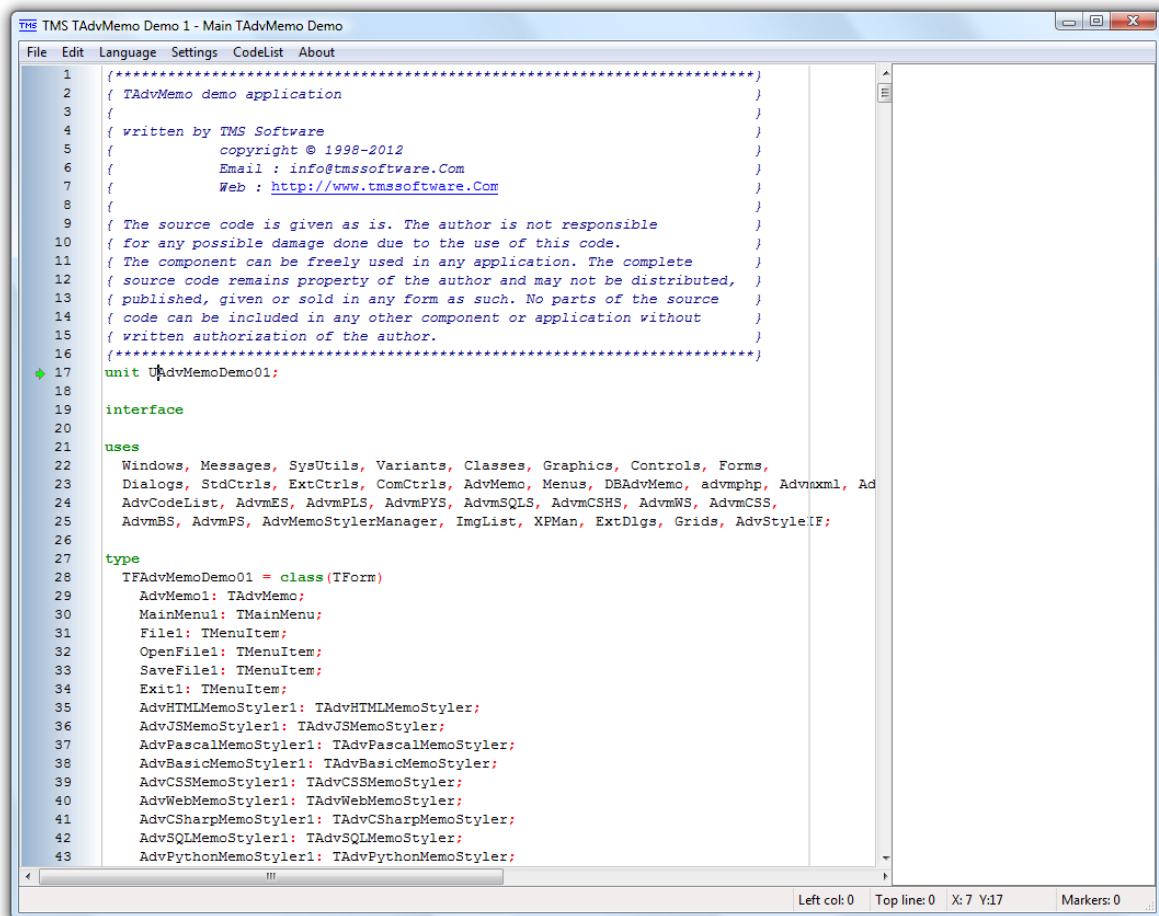
On the left side a TAdvMemo control is used, with gutter. On the right side, a TAdvCodeList is used. Selected code snippetsin the memo in the memocan be dragged and dropped from the TAdvMemo control to the TAdvCodeList, and the other way around. The event viewer can be activated in the edit menu.

I, various actions are offered for creating bookmarks, breakpoints, markers.Note that tThis allows you to see a log of the events as they are being triggered from the memo while in use.

Demo 2: Auto correction demo

Demo 2 shows the AutoCorrection functionality in TAdvMemo.

Some auto correction items are created automatically and displayed in a grid In this grid, nand new ones can be appended for testing.
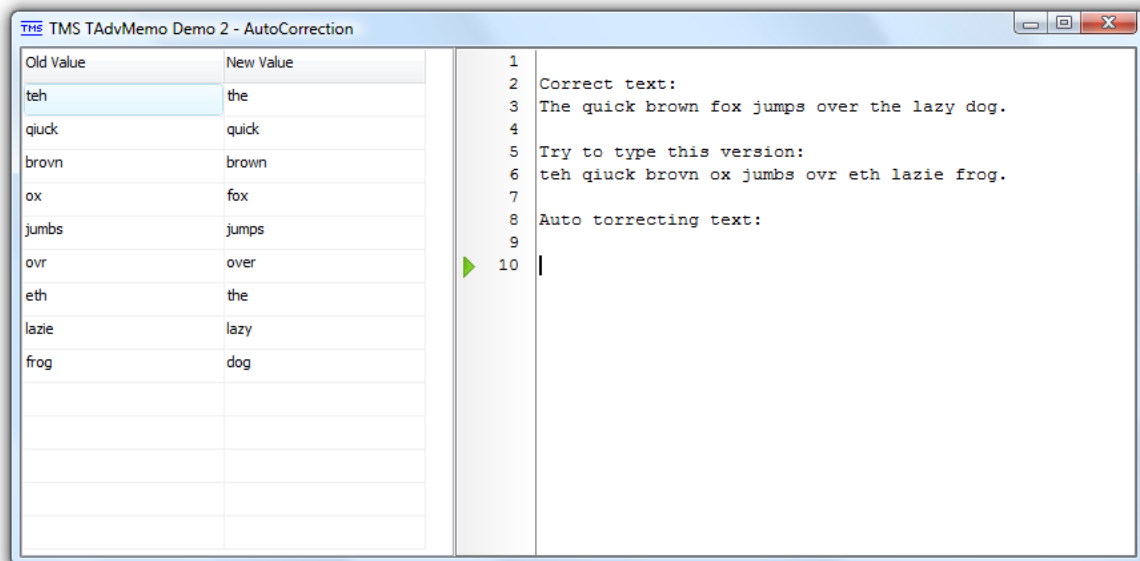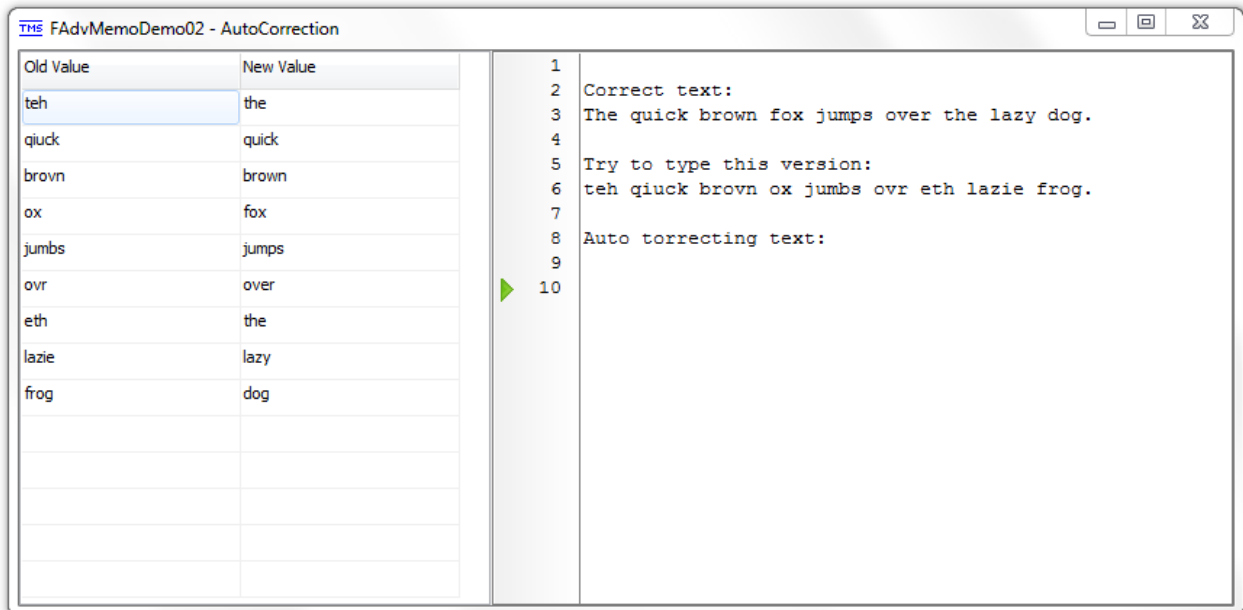
On the left side, the values that were created in program code. also automatically to ensure it is written with uppercaseA marker was created with a custom image, and used to set the cursor at the marker position.

Demo 3: Auto completion and drag & drop

Demo 3 demonstrates two of the functionalities: the auto completion functionality and the Drag&Drop between the TAdvCodelist and the TAdvMemo.
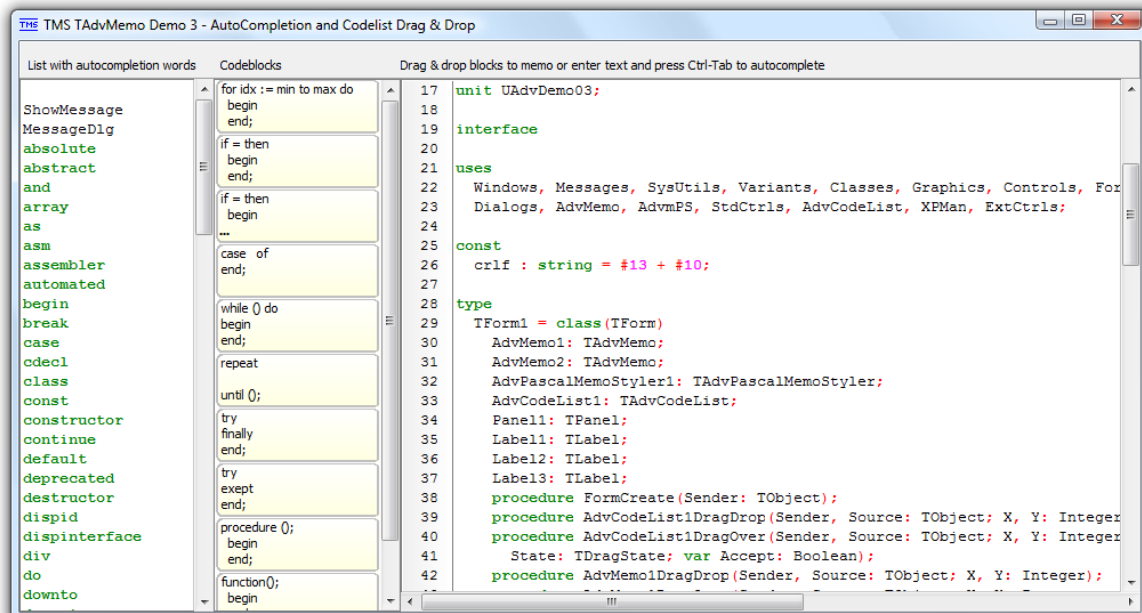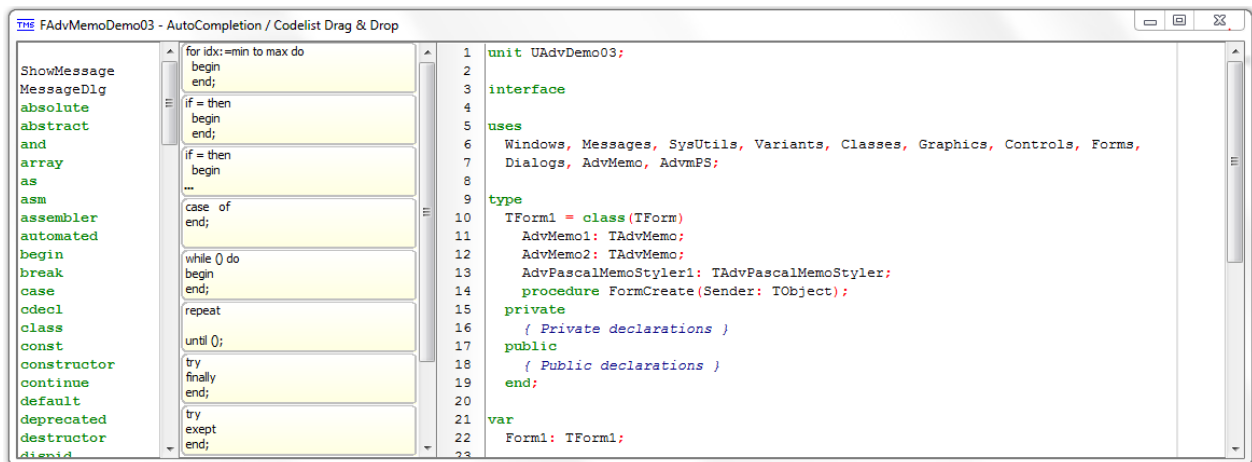
The form shows three main parts: at the left side, a TAdvMemo control without gutter just showing the keywords that will be completed. When pressing Ctrl + Space in the memo at the right side after typing a part of the string (that has to be completed), a screen will popup showing the different choices (taken from the memo at the left side)

The middle section shows a TAdvCodeList with programmatically created code blocks. These blocks can be dragged & dropped to the TAdvMemo on the right side. Selected code blocks from the right hand side TAdvMemo control can be dragged & dropped in the TAdvCodeList.

Demo 4: Codefolding

Demo 4 shows how to use regions for on code-folding in the TAdvMemo.
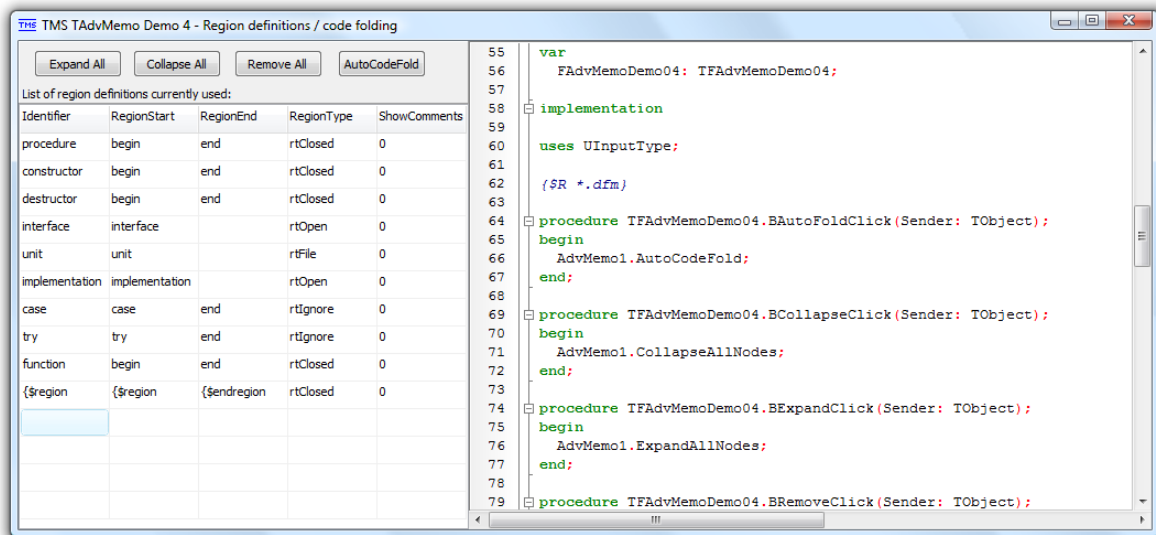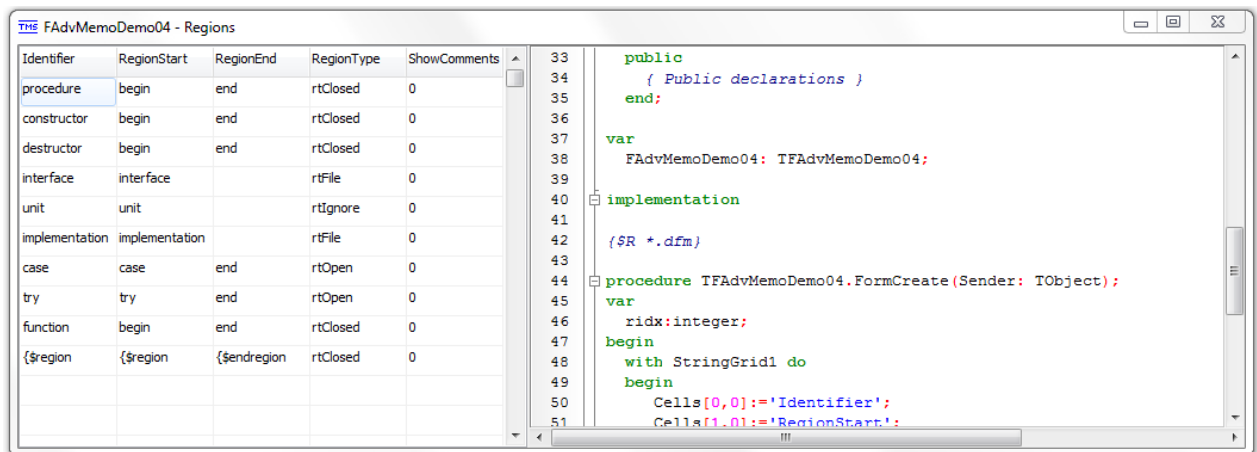
The region definitions are displayed in a grid on the left side. The region definition comprises a region start marker, region end marker and a region type. Region information can be modified in the grid. When pressing Enter, a new region will be created with 'New' as identifier. The TAdvMemo reflects these changes, clicking on the node folds the 'new' codeblock.nodeon the left side

Demo 5: Auto completion integration

Demo 5 shows to use the OnGetAutoCompletion event to integrate the auto completion with real object interfaces: properties, events, methods.
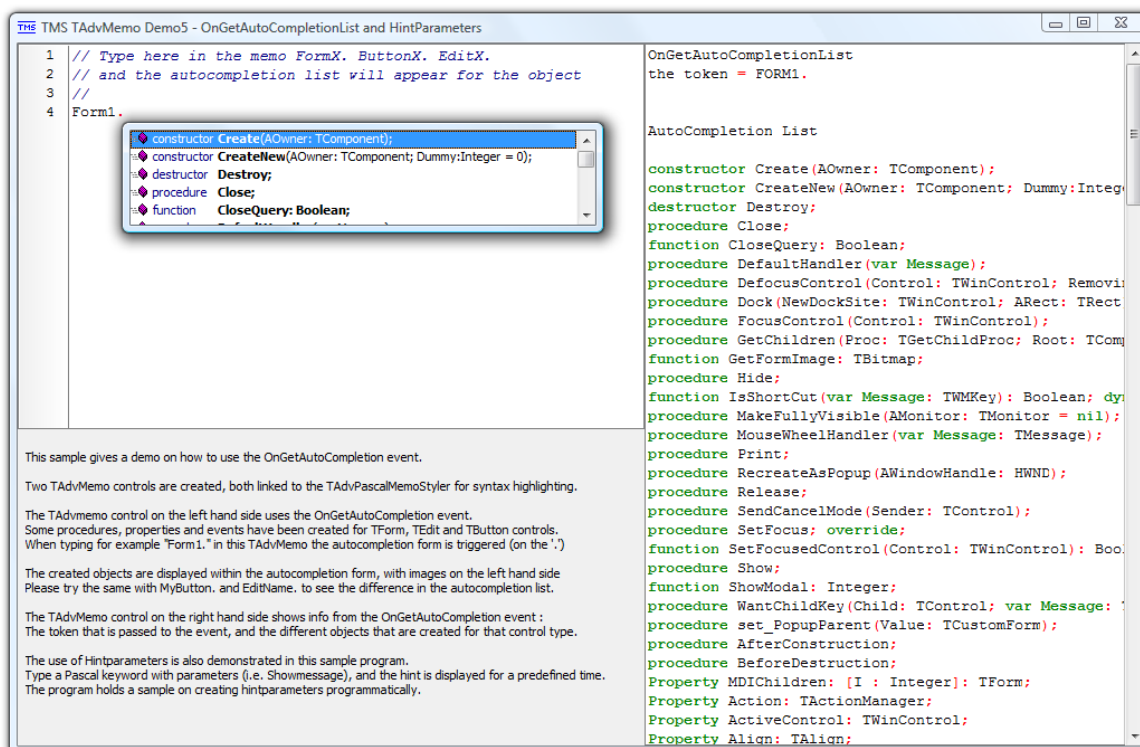
Two TAdvMemo controls are used, both linked to the TAdvPascalMemoStyler for syntax highlighting. The TAdvmemo control on the left side uses the OnGetAutoCompletion event. Some procedures, properties and events have been predefined in code for TForm, TEdit and TButton classes. When typing for example 'Form1.' in this TAdvMemo, the auto completion listbox will be shown (Auto completion start character is set to ''.''). The full interface of the sample objects are displayed within the auto completion listbox with images and marking the types properties, events, methods.

Try the same with 'MyButton.' and 'EditName.', this shows the interfaces for these classes in the auto completion list.

The TAdvMemo control on the right hand side shows the full information that was retrieved via the OnGetAutoCompletion event, the token that is passed to the event and based on the token, the full interface of the token class is returned in this event.

, t can be found M

Demo 6: Use of the emoticon styler shows TAdvEmoticonMemoStyler to show along
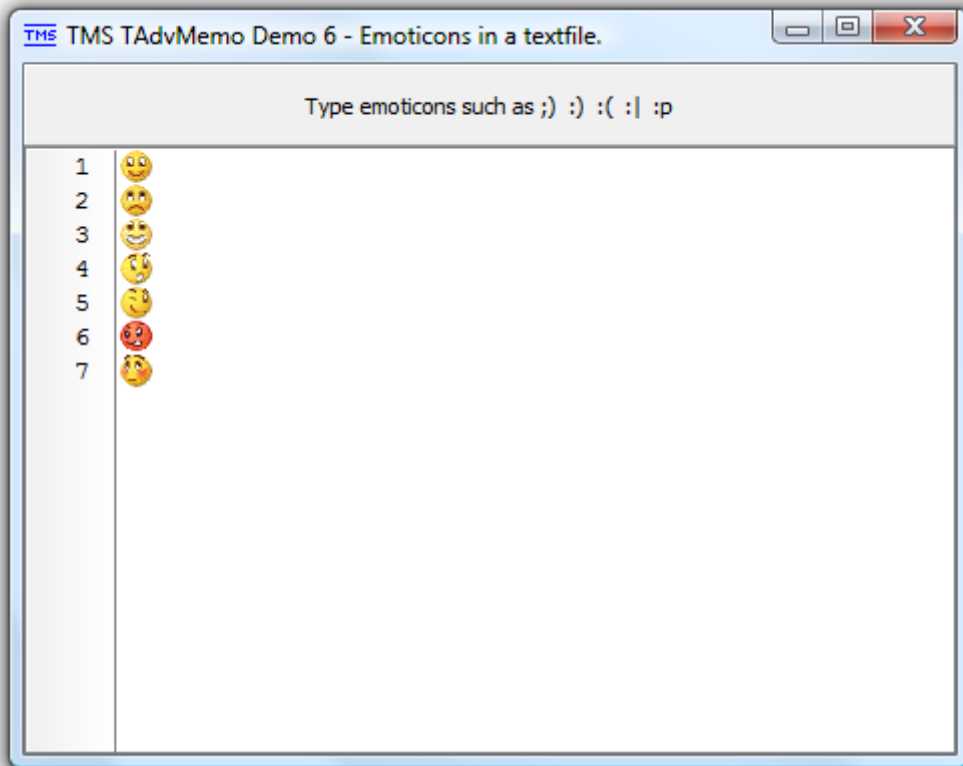
in

Demo 7: Use of the TAdvMemoSource to create multi file editorIn this demo, the TAdvMemoSource is used together with the TAdvMemo to create a multi file editor with a single TAdvMemo. The TAdvMemoSource instances are created when a new file is opened and serve to store the text of the memo. All TAdvMemoSource instances are added to a TObjectList. This code shows how the TObjectList is created and the first instance of TAdvMemoSource is added to it and finally assigned to the TAdvMemo:

```
begin

  SourceList := TObjectList.Create;
  SourceList.Add(TAdvMemoSource.Create(self));
  TAdvMemoSource(SourceList.Items[SourceIndex]).SyntaxStyler :=
AdvMemo1.SyntaxStyles;
  SourceIndex := 0;
  AdvMemo1.MemoSource := TAdvMemoSource(SourceList.Items[SourceIndex]);
end;
```

Switching between files is then as easy as just setting the AdvMemo.MemoSource property to point to the TAdvMemoSource for the new file:

```
procedure TForm1.TabControl1Change(Sender: TObject);
begin
  SourceIndex := TabControl1.TabIndex;
  AdvMemo1.MemoSource := TAdvMemoSource(SourceList.Items[SourceIndex]);
end;
```
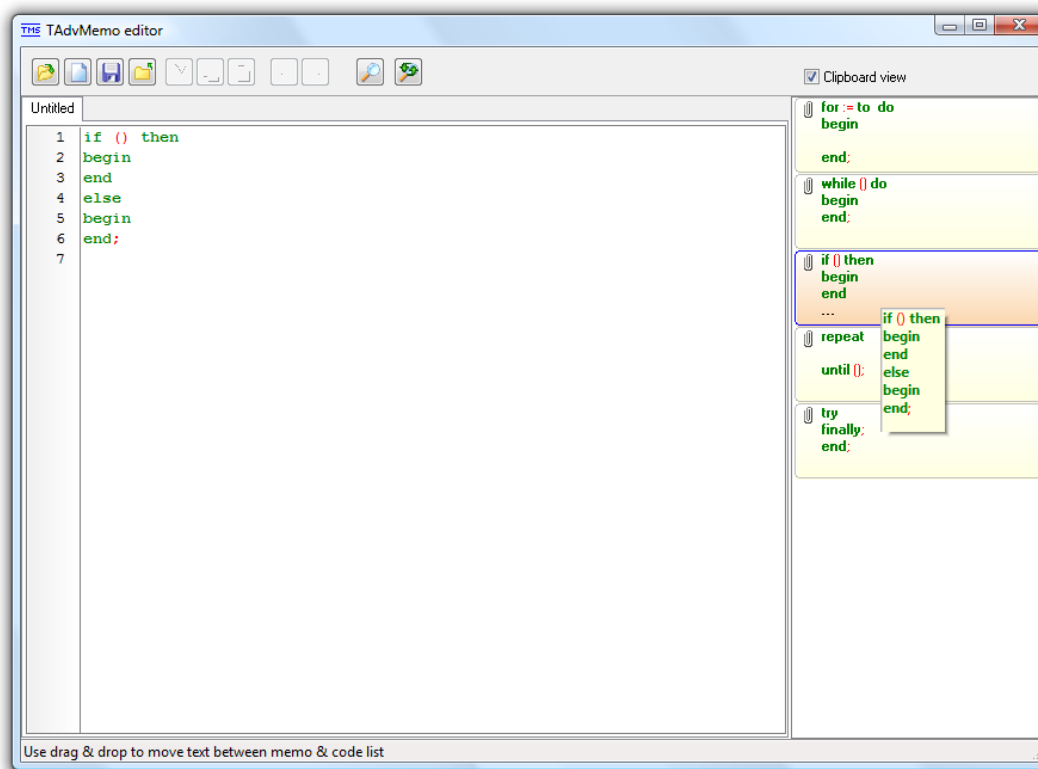

Demo 8: Common operations demoThis demo shows most common operations for the TAdvMemo. This shows selecting between 4 different syntax highlighting styles, undo/redo functions, find & replace, printing, HTML export as well as clipboard operations.
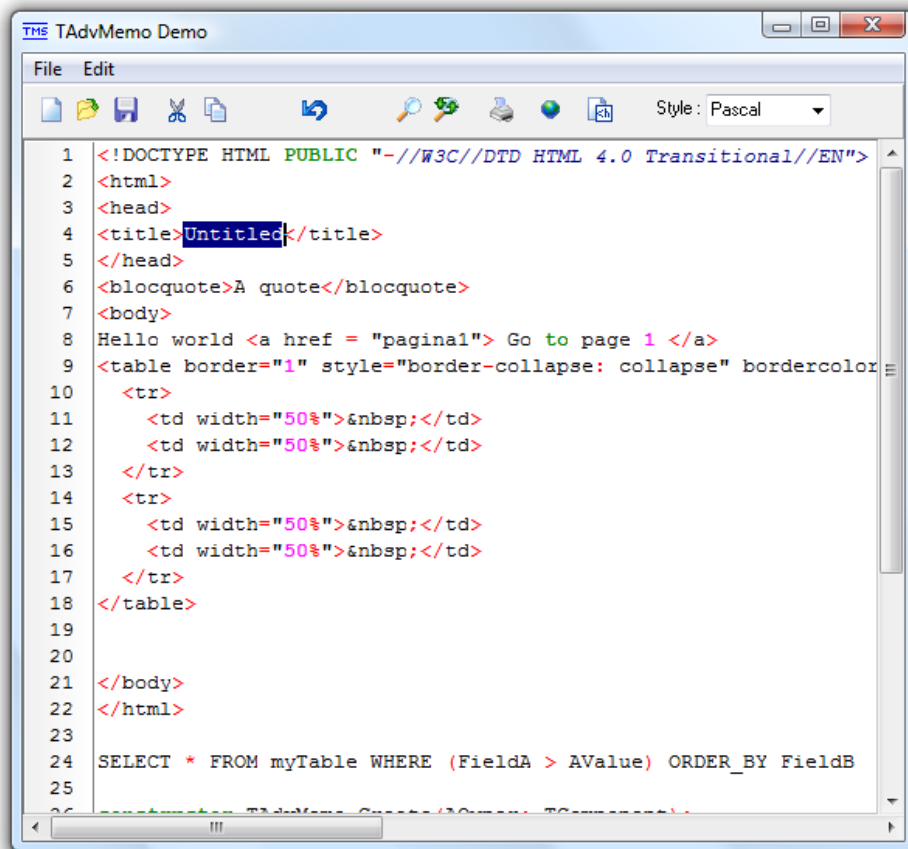
Demo 9: Addict spell checker supportNote that it is required to have Addict spell checker (see http://www.addictive-software.com) installed to be able to use this demo as well as the component TAdvMemoAddictChecker. To install this component TAdvMemoAddictChecker, add the unit AdvMAddict.pas to either the TAdvMemo package file or the TMS Component Package file or a new package file and compile & install this package.

The demo shows that the Addict spell checker is called as words are typed in the TAdvMemo and automatic spell check / spell correction can happen while typing or when invoked.