# ***dbSoft*** - TSyntaxMemo collection

The TSyntaxMemo collection consists of the following components

**TSyntaxMemo** A TMemo compatible text editor with interfaces for syntax highlighting. Emulates TMemo methods, properties and message support. Extends TMemo by providing unlimited undo/redo support, support for large text files (limited only by available memory).

**TSyntaxMemoParser** A lexical analyser non-visual component used by TSyntaxMemo to provide dynamic analysis of text within the TSyntaxMemo control. Fully configurable for a wide variety of text highlighting environments

**TDBSyntaxMemo** Data aware version of TSyntaxMemo.

---

**Installing TSyntaxMemo**

**Ordering TSyntaxMemo and contacting the author**

---

**Please note:** This help file is under construction at present. Some topics may be empty and some may contain incomplete information.
Last update to help file was on November 27 1997.

Parser1
Parser2
Parser3
Parser4
Parser5
Parser6

## ActiveParser property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property ActiveParser: longint;
```

**Description**
ActiveParser is used to determine which of the Parsern TSyntaxMemoParser components will be used to provide syntax highlighting effects to the text of the control. ActiveParser must be in the range 1 to 6 (inclusive). If no parser component is attached to the specified Parsern property then TSyntaxMemo will behave as a plain text editor with no highlighting applied.

**Remarks**

LineHasGlyph
RemoveLineGlyph
LineGlyphs
TGlyphIndex

## AddLineGlyph method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure AddLineGlyph(G: byte; L: longint);
```

**Description**
AddLineGlyph is used to add a specific glyph to a line. When the line is displayed in TSyntaxMemo, the gutter will show the new glyph. The image index being added should be passed as the G parameter with the line number in the L parameter (Lines are numbered starting from 1).

**Remarks**
Lines can only display one glyph. If a line has more than one glyph assigned, it will be the glyph with the highest index value that will be displayed.

## TSyntaxMemo custom messages index

See also

| Message ID | Summary of function |
|---|---|
| **SEM_SELECTION** | Get / Set current selection extents |
| **SEM_OPTIONS** | Get / Set current options |
| **SEM_REPLACESEL** | Replace current selection by given text |
| **SEM_IMPORTSEL** | Import text into current selection from a TStream instance |
| **SEM_EXPORTSEL** | Export currently selected text to a TStream instance |
| **SEM_MODIFIED** | Set / Get modified status of control |
| **SEM_TOPLINEINDEX** | Set / Get current index of top line of display |
| **SEM_LEFTINDENT** | Set / Get current value of left line indent |
| **SEM_FINDTEXT** | Find text in document |
| **SEM_REPLACETEXT** | Find and replace text in document |
| **SEM_GETTEXT** | Copy text to memory buffer |
| **SEM_CANREDO** | Can we redo an action ? |
| **SEM_REDO** | ReDo last undo action |
| **SEM_INDENT** | Indent selected line(s) |
| **SEM_UNDENT** | Undent selected line(s) |
| **SEM_REPARSE** | Re-parse text |

## AttachEditor method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
procedure AttachEditor(Ed: TSyntaxMemo);
```

**Description**
TSyntaxMemoParser components can be used to provide syntax highlighting to more than one TSyntaxMemo editor control. At design time it is only necessary to assign the Parsern property of the TSyntaxMemo controls to attach an editor to a parser.
At run-time, an editor may be attached to an editor either by explicit assignment to the Parser*n* property, or by use of the AttachEditor method.
AttachEditor ensures that an editor is updated when the parser settings change. If the editor is subsequently removed or destroyed TSyntaxMemoParser will take care of the removal of the editor from the UpdateList.

**Remarks**

## CaretPos property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`property CaretPos: TPoint;`

**Description**
CaretPos given the line and column of the current insertion point. To set a new caret position, assign a new value to CaretPos.

**Remarks**
CaretPos will limit the Line/Column values to the limits of the text in the control. Any attempt to set a column beyond the end of the line will place the caret at the end of the line. Any attempt to refer to a line beyond the last line will result in the last line being used.
The property is declared as a record type `TPoint`, thus assignments must be made using variables of type `TPoint`. Attempts to use `CaretPos.Y := Z` will cause a compile-time error, however `Z := CaretPos.Y` will compile without error.

## Clear method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure Clear;
```

**Description**
Clear deletes all text in the TSyntaxMemo control. The current undo stack will be emptied and the Modified property will be false after Clear returns.

**Remarks**

SetBookmark
IsBookmarkSet

## ClearBookmark method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure ClearBookmark(n: byte);
```

**Description**
ClearBookmark is used to remove a bookmark from the file being edited. The bookmarks are number 0 to 9 and should be passed as the parameter n. An exception will be raised if a value greater than 9 is passed.

**Remarks**
If the passed bookmark has not been set, no action will taken place. If it has been set it will be cleared and the display updated.

CompileFromStream
CompileScript
Script

## Compile property

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
Not applicable
```

**Description**
Compile is displayed in the Delphi IDE property inspector. It displays a single ellipsis (...) which, when pressed, will cause the text of the file specified by the Script property to be compiled.

**Remarks**
Compile is a design time only property. Use CompileScript at run-time.

Compile
CompileScript
Script

## CompileFromStream method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
procedure CompileFromStream(aStream: TStream);
```

**Description**
CompileFromStream allows scripts to be compiled at run-time from a supplied TStream instance.

**Remarks**
See CompileScript for details of the compilation process.

Compile
CompileFromStream
Script

## CompileScript method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
procedure CompileScript;
```

**Description**
CompileScript, at run-time, compiles the text in the file specified by the Script property. If compilation is successful then the syntax highlight specification given in the script will be applied to all attached editors (see UpdateEditors method).

**Remarks**
If, during the compilation process, an error is detected, the TSyntaxMemoParser will be left in its default state. This state applies no syntax highlighting to attached editors.
See script reference for details of errors reported by the script compiler

CursorTokenText
Script overview
OnHyperlinkHover event
OnHyperlinkClick event

## CursorToken property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`property CursorToken: byte;`

**Description**
CursorToken returns the token value of the syntax elemnet underneath the mouse cursor.

**Remarks**
The byte value returned is the token value before translation by the `%%map` section in the current parser's script. See script reference for details.

CursorToken
Script overview
OnHyperlinkHover event
OnHyperlinkClick event

# CursorTokenText property

See also		Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property CursorTokenText: string;
```

**Description**
CursorTokenText returns the actual text of the syntax element in which the mouse cursor is positioned.

**Remarks**
The text of the current syntax element can be any length, from a few characters to many lines of text. If the text of the syntax element is long then there may be a noticeable speed performance degredation if this property is read often. A check for the CursorToken should be made for relevance before requesting the actual text of the syntax element.

## def_BackColor property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property def_BackColor: TColor;
```

**Description**
def_BackColor allows the reading and setting of the default text color for a TSyntaxMemo control The default text color is used to fill in the background of the default sections of the file being edited in a TSyntaxMemo.

**Remarks**
def_BackColor is language dependent. For each possible value of the Language property a possibly different def_BackColor value may be stored.

## def_FontStyle property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`property def_FontStyle: TFontStyles`

**Description**
`def_FontStyle` is applied to syntax elements that are unrecognised by the TSyntaxMemoParser component specified by the ActiveParser property.

**Remarks**
Setting def_FontStyle should be done by assignment of a TFontStyles variable.

## def_TextColor property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property def_TextColor: TColor;
```

**Description**
def_TextColor allows the reading and setting of the default text color for a TSyntaxMemo control The default text color is used to fill in default text sections of the file being edited in a TSyntaxMemo.

**Remarks**
def_TextColor is language dependent. For each possible value of the Language property a possibly different def_TextColor value may be stored.

## DocTitle property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`property DocTitle: string;`

**Description**
DocTitle is used in the header of printed output when the Print method is invoked. By default it is assigned to the filename when LoadFromFile or SaveToFile is used to fill/save the control text. Prior to calling the Print method, the DocTitle property can be changed to show a different header text.

**Remarks**
DocTitle defaults to the empty string if LoadFromFile/ SaveToFile is never called.

## EBNF syntax specification

EBNF (Extended Backaus-Naur Form) is a style of specification used for formal syntax descriptions. Within the syntax description the following metacharacters are used:

| | |
|---|---|
| `a ::= b` | Construct **a** is defined by construct **b** |
| `[a]` | Indicates that construct **a** is optional |
| `(abc)` | Indicates that constructs **a**, **b** and **c** are taken as a single construct |
| `a\|b` | Indicates either construct **a** or construct **b** |
| `'abc'` | The literal characters 'a' followed by 'b' followed by 'c' |
| `<a>` | Single syntax construct **a** defined in the specification |

## Effect property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Effect[E: byte]: TFormatEntry;
```

**Description**
The Effect[] array gives access to the actual colors and font effects applied to each syntax token.
Accessing the Effects[] property of TSyntaxMemo will not cause the changes to be preserved in the registry or when the current parser is changed via the ActiveParser property.
Effect[] is designed to give quick access to the styles currently being applied.

**Remarks**
To set new properties use the methods of TSyntaxMemoParser.

## TSyntaxMemoParser events index

See also: Properties Methods Tasks

| Event | Type | Summary |
|---|---|---|
| **OnCustomParse** | TCustomParseEvent | Perform custom parsing functions in response to match(n) script statements |
| **OnModifyProperties** | TNotifyEvent | Property editor about to be displayed. |

## TSyntaxMemo events index

See also: Properties Methods Tasks

| Event | Type | Summary |
|---|---|---|
| **OnGutterClick** | TGutterClick | Mouse has been clicked in the left hand gutter area |
| **OnHyperlinkClick** | THyperEvent | Mouse has been clicked on a hotspot within the control |
| **OnHyperlinkHover** | TNotifyEvent | Mouse is over a hotspot within the control |
| **fn_CharIsInWord** | **TCharTest** | Test if a character can be in a word |
| **fn_CharStartsWord** | **TCharTest** | Test if a character can start a word |

## fn_CharIsInWord event

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property fn_CharIsInWord: TCharTest;
```

**Description**
TSyntaxMemo will generate TCharTest events whenever it is required to test a character. fn_CharIsInWord and fn_CharStartsWord are two events that TSyntaxMemo generates to allow applications to override the definition of characters that make up 'words'. TCharTest event handlers will be passed a single character to test and should return true if the condition is met, false otherwise.

**Remarks**
At present TSyntaxMemo will only generate events when selecting words (double click), moving to word boundaries (**Ctrl+Left Arrow** / **Ctrl+Right Arrow**). TSyntaxMemo may be upgraded in the future to allow further processing of words within files. These upgrades will use any attached event handlers to carry out the required checks.

## fn_CharStartsWord event

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`property fn_CharStartsWord: TCharTest;`

**Description**
TSyntaxMemo will generate TCharTest events whenever it is required to test a character. fn_CharIsInWord and fn_CharStartsWord are two events that TSyntaxMemo generates to allow applications to override the definition of characters that make up 'words'. TCharTest event handlers will be passed a single character to test and should return true if the condition is met, false otherwise.

**Remarks**
At present TSyntaxMemo will only generate events when selecting words (double click), moving to word boundaries (**Ctrl+Left Arrow** / **Ctrl+Right Arrow**). TSyntaxMemo may be upgraded in the future to allow further processing of words within files. These upgrades will use any attached event handlers to carry out the required checks.

## Gutter property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Gutter: longint;
```

**Description**
The Gutter property describes the width, in pixels, of the gray gutter at the left edge of the control display. The minimum, non-zero, width that Gutter can be set to is 34 pixels, the maximum is 20% of the width of the control. Setting Gutter to zero will cause the gray area not to be displayed.

**Remarks**
Gutter defaults to a value of 34 pixels.

## GutterGlyphs property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property GutterGlyphs: TImageList;
```

**Description**
GutterGlyphs are user defined 16x16 images that are displayed in the left gray gutter area of the control. Each 16x16 image in the supplied TImageList will be used by the AddLineGlyph, RemoveLineGlyph and the LineHasGlyph methods.

**Remarks**
Up to 16 images may be contained in each GutterImage.

## HyperCursor property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property HyperCursor: TCursor;
```

**Description**
HyperCursor is used when the mouse is over a syntax element that is declared as a 'hotspot'. The mouse cursor will change to the shape specified in the HyperCursor property.

**Remarks**
HyperCursor defaults to crIBeam, the normal mouse cursor within text in the TSyntaxMemo control.

## IndentStep property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property IndentStep: longint;
```

**Description**
IndentStep defines how many spaces will be inserted at the start of lines when the SEM_INDENT message is used.

**Remarks**
IndentStep defaults to a value of 1, indiacting that each selected line will be indented by one space by the SEM_INDENT message.

## InsertMode property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property InsertMode: Boolean;
```

**Description**
InsertMode returns the value of the insert status of the control. A value of TRUE indicates that new text will be inserted into the control, a value of FALSE indicates that new text will overrwrite existing text. Setting InsertMode to TRUE or FALSE will update the InsertMode appropriately.

**Remarks**

ClearBookmark
SetBookmark

## IsBookmarkSet method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
function IsBookmarkSet(n: byte; var Line, Col: longint): Boolean;
```

**Description**
IsBookmarkSet is used to determine if a given bookmark n has been set. If it has been set, the method returns TRUE and the Line and Col parameters will be set to the Line and Column respectivley of the bookmark. If the bookmark has not been set then the method will return FALSE and the values of Line and Col will be invalid.
Passing a bookmark number n outside the range 0 to 9 will cause an exception to be raised.

**Remarks**
The returned Line and Column values are one based. That is, the first line of a document is line number one and the leftmost column is column one. When a bookmark is set the Line and Column are stored. IsBookmarkSet will return the highest legal column value for the bookmark, if the bookmarked line has less that the original bookmarked columns then the parameter Col will be set to the length of the line. Should the line exceed the bookmarked column width then the bookmarked column value will be returned.

## JumpToBookmark method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure JumpToBookmark(n: byte);
```

**Description**
JumpToBookmark will move the caret to the stored location of the given bookmark n.
Passing a bookmark number n outside the range 0 to 9 will cause an exception to be raised.

**Remarks**
When a bookmark is set the Line and Column are stored. JumpToBookmark will move to the highest legal column value for the bookmark, if the bookmarked line has less that the original bookmarked columns then the caret will be placed at the end of the line. Should the line exceed the bookmarked column width then the caret will be placed at the original bookmarked column position.

## Supported keyboard commands

TSyntaxMemo supports a default keyboard interface. This can be enabled or disabled by use of the Options property, the default is for all options to be enabled. The keyboard commands are grouped by function and enabling options below:

| **Caret movement** | **Enabled when option flag** `eo_KEYS_MOVE` **is set** |
|---|---|
| **End** | Moves to the end of a line |
| **Home** | Moves to the start of a line |
| **Enter** | Inserts a carriage return |
| **Ins** | Turns insert mode on/off |
| **Del** | Deletes the character to the right of the caret |
| **Backspace** | Deletes the character to the left of the caret |
| **Space** | Inserts a blank space |
| **Left Arrow** | Moves the caret left one character |
| **Right Arrow** | Moves the caret right one character |
| **Up Arrow** | Moves the caret up one line |
| **Down Arrow** | Moves the caret down one line |
| **PgUp** | Moves the display up one page |
| **PgDn** | Moves the display down one page |
| **Ctrl+Left Arrow** | Moves one word left |
| **Ctrl+Right Arrow** | Moves one word right |
| **Ctrl+Home** | Moves to start of file |
| **Ctrl+End** | Moves to end of file |

| **Text selection** | **Enabled when option flag** `eo_KEYS_SELECT` **is set** |
|---|---|
| **Shift+Left Arrow** | Selects the character to the left of the caret |
| **Shift+Right Arrow** | Selects the character to the right of the caret |
| **Shift+Up Arrow** | Moves the caret up one line and selects from the left of the starting caret position |
| **Shift+Down Arrow** | Moves the caret down one line and selects from the right of the starting caret position |
| **Shift+PgUp** | Moves the caret up one screen and selects from the left of the starting caret position |
| **Shift+PgDn** | Moves the caret down one screen and selects from the right of the starting caret position |
| **Shift+End** | Selects from the caret position to the end of the current line |
| **Shift+Home** | Selects from the caret position to the start of the current line |
| **Ctrl+Shift+Left Arrow** | Selects from the current caret position to the start of the previous word |
| **Ctrl+Shift+Right Arrow** | Selects from the current caret position to the start of the next word |
| **Ctrl+Shift+Home** | Selects from the current caret position to the start of the current file |
| **Ctrl+Shift+End** | Selects from the current caret position to the end of the current file |

| **Clipboard functions** | **Enabled when option flag** `eo_KEYS_CLIPBOARD` **is set** |
|---|---|
| **CTRL+C** | Copy selected text to clipboard |
| **CTRL+X** | Cut selected text to clipboard |
| **CTRL+V** | Replace selected text with clipboard contents |

| **Miscellaneous functions** | **Enabled when option flag** `eo_KEYS_FUNCTIONS` **is set** |
|---|---|
| **CTRL+K+0** | Sets bookmark 0 |
| **CTRL+K+1** | Sets bookmark 1 |
| **CTRL+K+2** | Sets bookmark 2 |
| **CTRL+K+3** | Sets bookmark 3 |
| **CTRL+K+4** | Sets bookmark 4 |
| **CTRL+K+5** | Sets bookmark 5 |
| **CTRL+K+6** | Sets bookmark 6 |
| **CTRL+K+7** | Sets bookmark 7 |
| **CTRL+K+8** | Sets bookmark 8 |
| **CTRL+K+9** | Sets bookmark 9 |
| **CTRL+K+Ctrl+0** | Sets bookmark 0 |

| | |
|---|---|
| **CTRL+K+Ctrl+1** | Sets bookmark 1 |
| **CTRL+K+Ctrl+2** | Sets bookmark 2 |
| **CTRL+K+Ctrl+3** | Sets bookmark 3 |
| **CTRL+K+Ctrl+4** | Sets bookmark 4 |
| **CTRL+K+Ctrl+5** | Sets bookmark 5 |
| **CTRL+K+Ctrl+6** | Sets bookmark 6 |
| **CTRL+K+Ctrl+7** | Sets bookmark 7 |
| **CTRL+K+Ctrl+8** | Sets bookmark 8 |
| **CTRL+K+Ctrl+9** | Sets bookmark 9 |
| | |
| **CTRL+Q+0** | Jumps to bookmark 0 |
| **CTRL+Q+1** | Jumps to bookmark 1 |
| **CTRL+Q+2** | Jumps to bookmark 2 |
| **CTRL+Q+3** | Jumps to bookmark 3 |
| **CTRL+Q+4** | Jumps to bookmark 4 |
| **CTRL+Q+5** | Jumps to bookmark 5 |
| **CTRL+Q+6** | Jumps to bookmark 6 |
| **CTRL+Q+7** | Jumps to bookmark 7 |
| **CTRL+Q+8** | Jumps to bookmark 8 |
| **CTRL+Q+9** | Jumps to bookmark 9 |
| **CTRL+Q+Ctrl+0** | Jumps to bookmark 0 |
| **CTRL+Q+Ctrl+1** | Jumps to bookmark 1 |
| **CTRL+Q+Ctrl+2** | Jumps to bookmark 2 |
| **CTRL+Q+Ctrl+3** | Jumps to bookmark 3 |
| **CTRL+Q+Ctrl+4** | Jumps to bookmark 4 |
| **CTRL+Q+Ctrl+5** | Jumps to bookmark 5 |
| **CTRL+Q+Ctrl+6** | Jumps to bookmark 6 |
| **CTRL+Q+Ctrl+7** | Jumps to bookmark 7 |
| **CTRL+Q+Ctrl+8** | Jumps to bookmark 8 |
| **CTRL+Q+Ctrl+9** | Jumps to bookmark 9 |
| | |
| **CTRL+K+I** | Indents selected text by one space |
| **CTRL+K+U** | Unindents selected text by one space |

## LanguageNames property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property LanguageNames: TStringList;
```

**Description**
A read-only property that lists the names of the editing environments provided by the attached TSyntaxMemoParser controls. The strings in the returned TStringList are taken from the script used by each parser (%%language Name section).

**Remarks**

## LastFindText property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property LastFindText: string;
```

**Description**
LastFindText returns the text used in the last find text operation performed by the control.

**Remarks**
If no text has been searched for then LastFindText will be the empty string.

## LineColor property

See also		Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property LineColor[aLine: longint]: TColor;
```

**Description**
The LineColor property of lines within the control allows the background color of a line to be overriden. In conjunction with the LineTextColor property lines can be highlighted as a whole rather than just the syntactic elements within. A common use for these properties is the visual indication of 'special' lines such as breakpoints, error lines etc.

**Remarks**
To remove any special display from the line, set the LineColor[] value to -1, any other TColor value will cause the line to be displayed with the current pallette.
The aLine index value should be a value within the range of lines in the control (use Lines.Count to obtain the number of lines). If the passed value is in excess of the number of lines then the last line will be affected.

## LineGlyphs property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property LineGlyphs[aLine: longint]: TGlyphIndex;
```

**Description**
LineGlyphs allows the glyphs of a line to be set as a whole. Each bit in the value represents the corresponding bitmap image in the GutterGlyphs TImageList property. Bit zero refers to image zero, bit 1 to image 1 etc. A maximum of 16 glyphs may be specified for a line and the glyph with the highest bit set will be displayed.
If a bit is on (i.e. a value of one) then the glyph is included in the line, if the bit is off (i.e. a value of zero) then the relevant glyph is not displayed.

**Remarks**
The aLine index value should be a value within the range of lines in the control (use Lines.Count to obtain the number of lines). If the passed value is in excess of the number of lines then the last line will be affected.

## LineHasGlyph method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
function LineHasGlyph(G: byte; L: longint): Boolean;
```

**Description**
LineHasGlyph is used to test whether a line within a TSyntaxMemo control will display a particular Glyph image in the gutter. The image index being tested should be passed as the G parameter with the line number in the L parameter (Lines are numbered starting from 1). If the line is present and it is currently displaying the passed glyph image then the result will be true otherwise the result will be false.

**Remarks**
Lines can only display one glyph. If a line has more than one glyph assigned, it will be the glyph with the highest index value that will be displayed.
The L index value should be a value within the range of lines in the control (use Lines.Count to obtain the number of lines). If the passed value is in excess of the number of lines then the last line will be affected.

## LineTextColor property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property LineTextColor[aLine: longint]: TColor;
```

**Description**
The LineTextColor property of lines within the control allows the text color of a line to be overriden. In conjunction with the LineColor property lines can be highlighted as a whole rather than just the syntactic elements within. A common use for these properties is the visual indication of 'special' lines such as breakpoints, error lines etc.

**Remarks**
To remove any special display from the line, set the LineTextColor[] value to -1, any other TColor value will cause the line to be displayed with the current pallette.
The aLine index value should be a value within the range of lines in the control (use Lines.Count to obtain the number of lines). If the passed value is in excess of the number of lines then the last line will be affected.

## LoadFromFile method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure LoadFromFile(Filename: string);
```

**Description**
The LoadFromFile method sets the components text to the contents of the passed filename. The current syntax effects of the ActiveParser property will be used to highlight the file once it has been loaded.

**Remarks**
After a new file has been loaded by the LoadFromFile or LoadFromStream methods, the undo stack will be cleared and the Modified will be false. The caret will be displayed at the start of the new file.

## LoadFromStream method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure LoadFromStream(aStream: TStream);
```

**Description**
The LoadFromStream method sets the components text to the text obtained from the passed stream instance. The current syntax effects of the ActiveParser property will be used to highlight the file once it has been loaded.

**Remarks**
After a new file has been loaded by the LoadFromFile or LoadFromStream methods, the undo stack will be cleared and Modified will be false. The caret will be displayed at the start of the new file.

## Map property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Map[E: byte]: byte;
```

**Description**
The Map table contains the mapping of syntax token values to the applied syntax effects values. E should be a syntax token value and the result will be the entry in the %%effects section of the script that will be applied to the syntax token.

**Remarks**
The Map[] property of TSyntaxMemo is designed to give quick access the the currently applied styles. To alter the mapping applied to a syntax token, the TSyntaxMemoParser TokenTable property should be accessed and the changes reflected to the TSyntaxMemo controls via the UpdateEditors method of TSyntaxMemoParser. In this way changes made to the mapping table will be preserved when the parser changes and the settings may be optionally stored in the registry.

## Margin property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Margin: byte;
```

**Description**
Margin is used to specify an spacing between the gutter and the left edge of text displayed in the control. This area will be colored in the default background of the TSyntaxMemoParser component enabled by the ActiveParser property.

**Remarks**
The default value for Margin is 2 pixels. Margin may be set to a pixel value up to the width of one character (in pixels).

## TSyntaxMemo methods index

See also: Properties Events Tasks

| Method | Summary |
|---|---|
| **Clear** | Clear contents |
| **LoadFromFile(Filename: string)** | Load a file into control using current parser |
| **LoadFromStream(F: TStream)** | Load contents of control from a stream |
| **SaveToFile(Filename: string)** | Save contents of control to a file |
| **SaveToStream(F: TStream)** | Save contents of control to a stream |
| **Print** | Print contents of control using current highlighting and printing options |
| **ModifyProperties** | Invoke property editor to alter highlighting effects |
| **AddLineGlyph(G: TGlyphIndex; L: longint)** | Display glyph with a given line of text |
| **RemoveLineGlyph(G: TGlyphIndex; L: longint)** | Remove glyph from a given line of text |
| **LineHasGlyph(G: TGlyphIndex; L: longint)** | Is a line displaying a given glyph ? |
| **JumpToBookmark(n: byte)** | Move caret to bookmarked line and column |
| **SetBookmark(n: byte; Line, Col: longint)** | Remember a given line and column as a bookmark |
| **IsBookmarkSet(n: byte; var Line, Col: longint)** | Has this bookmark been set before ? |
| **ClearBookmark(n: byte)** | Remove bookmark setting |

## TSyntaxMemoParser method index

See also: Properties Events Tasks

| Method | Summary |
|---|---|
| **AttachEditor(Ed: TSyntaxMemo)** | Notify parser that a TSyntaxMemo control is using this parser |
| **CompileScript** | Compile file in Script property, replacing current internal parser model |
| **CompileFromStream(aStream: TStream)** | Compile script in supplied stream, replacing current internal parser model |
| **StylesAsString** | Get current settings as a formatted string value |
| **StylesFromRegistry**(UseDefault: Boolean; aKey: string) | Set settings from registry entry (= RegistryKey if UseDefault is true) |
| **StylesFromString**(Styles: string) | Set settings from a formatted string |
| **StylesToRegistry** | Save settings in registry using RegistryKey |
| **UpdateEditors** | Update attached editor(s) with current settings |

## ModifyProperties method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure ModifyProperties;
```

**Description**
Invoke the **ModifyProperties** method of a **TSyntaxMemo** control to allow the user to alter the current syntax effects and styles. By default the TSyntaxMemoParser control currently in use by the **TSyntaxMemo** control (see ActiveParser) will display its property editor. This behaviour may be overriden in the **TSyntaxMemoParser** by providing an application property editor via the TSyntaxMemoParser's OnModifyProperties event handler.

**Remarks**

## OnCustomParse event

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
property OnCustomParse: TCustomParseEvent;
type TCustomParseEvent =
        function (Sender : TObject;
                  ParseID: longint;
                  IStream: TEdStream;
              var kLength, kValue: longint): Boolean of object;
```

**Description**
Within scripts of TSyntaxMemoParser controls, the Match(n) script statement causes an OnCustomParse event to be generated. This event is passed the TSyntaxMemoParser instance (**Sender**), the value n in the script (**ParseID**), a read-only TStream descendant (**IStream**) that allows access to the text of the TSyntaxMemo control.
OnCustomParse event handlers should return the length of the syntax element they recognise (**kLength**) and the token value for the recognised syntax element (**kValue**).
If a token is recognised then the event handler should return **TRUE**, otherwise it should return **FALSE**.

**Remarks**

## OnGutterClick event

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property OnGutterClick: TGutterClick;
```

**Description**
The OnGutterClick event occurs when the user clicks in the left gutter of a TSyntaxMemo control.

## OnHyperlinkClick event

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property OnHyperlinkClick: THyperEvent;
type THyperEvent =
      procedure(Sender   : TObject;
                HyperData: string;
                HyperType: longint) of object;
```

**Description**
Scripts of a TSyntaxMemoParser control may specify syntax elements as 'hotspots'. When the mouse is clicked within such a 'hotspot' syntax element a OnHyperlinkClick event is generated. The TSyntaxMemo instance that generated the event is passed (**Sender**) along with the text of the 'hotspot' syntax element (**HyperData**) and the syntax element token value (**HyperType**).

**Remarks**

## OnHyperlinkHover event

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`property OnHyperlinkHover: TNotifyEvent`

**Description**
Scripts of a TSyntaxMemoParser control may specify syntax elements as 'hotspots'. When the mouse is within such a 'hotspot' syntax element a OnHyperlinkHover event is generated. The CursorToken property can be used to determine the type of the syntax element.

**Remarks**

## OnModifyProperties event

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
property OnModifyProperties: TNotifyEvent
```

**Description**
The OnModifyProperties event is generated when the ModifyProperties method of TSyntaxMemoParser is invoked. OnModifyProperties event handlers may thus implement their own property editors. If no event handler is attached then the TSyntaxMemoParser default property editor will be used.

**Remarks**

## Options property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Options: TSyntaxMemoOptions;
```

**Description**
Options allows control of the various settings for a TSyntaxMemo. Each option can be used as below:

| | |
|---|---|
| **smoKeysMove** | Key assignments relating to caret movement |
| **smoKeysSelect** | Key assignments relating to text selection |
| **smoKeysClipboard** | Key assignments relating to clipboard operations |
| **smoKeysFunctions** | Key assignments relating to micellaneous functions |
| **smoPrintWrap** | When printing, wrap long lines |
| **smoPrintLineNos** | When printing, print line numbers |
| **smoPrintFilename** | When printing, print source filename in header |
| **smoPrintDate** | When printing, print current date/time in header |
| **smoPrintPageNos** | When printing, print page numbers in footer |
| **smoWordWrap** | Word-wrap text display |

The default setting is:
```
[smoKeysMove,  smoKeysSelect,   smoKeysClipboard, smoKeysFunctions,
 smoPrintWrap, smoPrintLineNos, smoPrintFilename, smoPrintDate,
smoPrintPageNos]
```

The display will be updated, where relevant, when the options settings are changed.

## Ordering TSyntaxMemo

TSyntaxMemo can be purchased from the author. The package includes the following:

**Full source code,**
**e-mail support direct from the author,**
**Free updates**

The purchase price of TSyntaxMemo is **49 UK Pounds**. Payment may be by one of the following methods:

**Pre-paid bankers cheque, or**
**Cash, or**
**Eurocheque, or**
**Direct bank to bank transfer**

If using any curreny other than UK Pounds, please include a 3 UK Pounds conversion fee. Please contact the author to arrange bank-to-bank transfers. Orders should be posted to the following address:

**David Brock**
**26 Kemp Avenue**
**Paisley**
**Scotland**
**United Kingdom**
**PA3 4JS**

The author can be contacted via e-mail at:
`dbrock@cqm.co.uk`

or via the TSyntaxMemo Web site at:
`http://users.colloquium.co.uk/~dbrock/synmemo/synmemo.htm`

## Parser1 property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Parser1: TSyntaxMemoParser;
```

**Description**
Parser1 is the TSyntaxMemoParser control that will be used to analyse the text of the TSyntaxMemo control and apply the highlighting effects of the script of that parser. To enable Parser1, the TSyntaxMemo ActiveParser property should be set to a value of 1.

**Remarks**

## Parser2 property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Parser2: TSyntaxMemoParser;
```

**Description**
Parser2 is the TSyntaxMemoParser control that will be used to analyse the text of the TSyntaxMemo control and apply the highlighting effects of the script of that parser. To enable Parser2, the TSyntaxMemo ActiveParser property should be set to a value of 2.

**Remarks**

## Parser3 property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Parser3: TSyntaxMemoParser;
```

**Description**
Parser3 is the TSyntaxMemoParser control that will be used to analyse the text of the TSyntaxMemo control and apply the highlighting effects of the script of that parser. To enable Parser3, the TSyntaxMemo ActiveParser property should be set to a value of 3.

**Remarks**

## Parser4 property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Parser4: TSyntaxMemoParser;
```

**Description**
Parser4 is the TSyntaxMemoParser control that will be used to analyse the text of the TSyntaxMemo control and apply the highlighting effects of the script of that parser. To enable Parser1, the TSyntaxMemo ActiveParser property should be set to a value of 4.

**Remarks**

## Parser5 property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Parser5: TSyntaxMemoParser;
```

**Description**
Parser5 is the TSyntaxMemoParser control that will be used to analyse the text of the TSyntaxMemo control and apply the highlighting effects of the script of that parser. To enable Parser5, the TSyntaxMemo ActiveParser property should be set to a value of 5.

**Remarks**

## Parser6 property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property Parser6: TSyntaxMemoParser;
```

**Description**
Parser6 is the TSyntaxMemoParser control that will be used to analyse the text of the TSyntaxMemo control and apply the highlighting effects of the script of that parser. To enable Parser6, the TSyntaxMemo ActiveParser property should be set to a value of 6.

**Remarks**

## Print method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure Print;
```

**Description**
Invoking the Print method will cause the contents of the control to be printed with the current printing options applied. The paper size and printer used will be determined by the current Windows settings. Syntax highlighting effects will be printed as on-screen.

**Remarks**

## TSyntaxMemoParser property index

See also: Methods Events Tasks

| Property | Type | Access | Summary |
|---|---|---|---|
| **AutoUpdateScript** | Boolean | read/write | Re-compile Script, if present, at run-time |
| **Compile** | n/a | n/a | Force design time compilation of script |
| **RegistryKey** | string | read/write | Registry base key for storage of editor settings |
| **Script** | string | read/write | Get/set script used to define parser behaviour |
| **UseRegistry** | Boolean | read/write | Enable/disable use of RegistryKey for settings storage |

## AutoUpdateScript property

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
`property AutoUpdateScript: Boolean;`

**Description**
At run-time, causes the TSyntaxMemoParser to look for the Script file. If located the file will be re-compiled and installed defining the new properties of the attached editor(s).

**Remarks**
At design-time, the properties of the TSyntaxMemo parser may be altered using the Delphi object inspector. These changes will be reflected in the Delphi IDE. At run-time the changes made at design time will be preserved unless overriden by the AutoUpdateScript property.
Use of AutoUpdateScript set to true is recommended in distributed applications when it allows upgrades to be shipped by the provision of a new script. In development of applications, AutoUpdateScript should be set to false to allow the iterative design process to be carried out. Only when the design is complete should the default script be updated with the desired changes and then the AutoUpdateScript property set to false.

## TSyntaxMemo properties index

See also: Methods Events Tasks

In addition to the properties of TMemo, TSyntaxMemo implements the following properties.
▶ indicates a run-time only property.

| | Property | Type | Access | Summary |
|---|---|---|---|---|
| | **ActiveParser** | longint | read/write | Get/set current parser index |
| ▶ | **CaretPos** | TPoint | read/write | Get/set current line and column of caret |
| | **CursorTokenText** | string | read only | Get text of token underneath mouse cursor |
| | **CursorToken** | byte | read only | Get value of token under mouse cursor |
| | **def_TextColor** | TColor | read/write | Get/set current default text color |
| | **def_FontStyle** | TFontStyles | read/write | Get/set current default font effects |
| | **def_BackColor** | TColor | read/write | Get/set current default background color |
| ▶ | **DocTitle** | string | read/write | Get/set current document title |
| ▶ | **Effect[E: byte]** | TformatEntry | read/write | Get/set effect style for defined syntax token |
| ▶ | **HyperCursor** | TCursor | read/write | Get/set cursor used when mouse over hotspot |
| | **Gutter** | longint | read/write | Get/set current gutter width in pixels |
| ▶ | **GutterGlyphs** | TimageList | read/write | Get/set images used for line margin |
| | **IndentStep** | byte | read/write | Get/set block indent/unindent amount in characters |
| | **InsertMode** | Boolean | read/write | Get/set insert (true) or overwrite (false) mode |
| ▶ | **LanguageNames** | TStringList | **read only** | Get list of environments implemented by attached parsers |
| | **LastFindText** | string | **read only** | Get last string used as find text parameter |
| ▶ | **LineGlyphs[L: longint]** | **TGlyphIndex** | read/write | Get/set current glyphs displayed by a given line |
| ▶ | **LineColor[L: longint]** | TColor | read/write | Get/set line override background color |
| ▶ | **LineTextColor[L: longint]** | TColor | read/write | Get/set line override text color |
| | **Margin** | byte | read/write | Get/set gap between gutter and text |
| ▶ | **Map[E: byte]** | byte | read/write | Get/set token to effect map table setting |
| | **Options** | TSyntaxMemoOptions | read/write | Get/set editor options |
| | **Parser1** | TCustomSyntaxMemoParser | read/write | Get/set parser of index value 1 |
| | **Parser2** | TCustomSyntaxMemoParser | read/write | Get/set parser of index value 1 |
| | **Parser3** | TCustomSyntaxMemoParser | read/write | Get/set parser of index value 1 |
| | **Parser4** | TCustomSyntaxMemoParser | read/write | Get/set parser of index value 1 |
| | **Parser5** | TCustomSyntaxMemoParser | read/write | Get/set parser of index value 1 |
| | **Parser6** | TCustomSyntaxMemoParser | read/write | Get/set parser of index value 1 |
| | **SaveFormat** | TSaveFormat | read/write | Get/set SaveToFile format |
| | **WrapAtColumn** | word | read/write | Get/set column used for word-wrapping |
| | **WrapOverride** | string | read/write | Get/set characters that override word-wrapping at start of lines |

## SaveFormat property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property SaveFormat: TSaveFormat;

type TSaveFormat = (sfTEXT, sfRTF);
```

**Description**
The SaveFormat property is used to determine in which format the text of the editor control will be saved by the SaveToFile and SaveToStream methods of TSyntaxMemo. The default is sfTEXT which causes the text to be saved as plain CR/LF terminated ASCII text. Setting SaveFormat to sfRTF will cause the contents of the control to be saved as Rich Text Format with all font, color and layout data intact.

**Remarks**
When copying text to the clipboard, TSyntaxMemo will place a RTF version of the selected text in addition to a plain text version.

## Purpose of TSyntaxMemoParser scripts

Scripts allow users to either customise an existing editing environment at design time or to carry out such actions at run-time within an application.

Accessing the TSyntaxMemo Web site (or contacting the author) will give access to a set of pre-written scripts for HTML, C, Object Pascal and an e-mail editing environment. Each of these has been created through the use of scripts and these scripts offer the best illustration of what can be done and how to achieve it.

Scripting is assisted by a pre-processor built into the TSyntaxMemoParser compiler, this allows users to write scripts that have meaning for them and aid in the maintainablity of scripts by different authors. Little or no restrictions have been placed on the length or complexity of scripts and users should find the environment capable of achieving most tasks.

The structure of scripts is given in both an informal descriptive manner and formal specification is provided.

Scripts have limitations and these are detailed and possible solutions presented.

## RegistryKey property

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
property RegistryKey: string;
```

**Description**
TSyntaxMemoParser controls may save their current style settings to the registry. The exact location in the registry is formed from the RegistryKey property followed by the Name in the `%%language` section of the script used by the TSyntaxMemoParser in its Script property. As an example, if a scripts `%%language` section contains:
```
   %%language
   Name = 'Object Pascal'
```
And the RegistryKey property is set to `'MyApp'` then the settings for the parser will be stored in the registry at location:
```
   'MyApp\Highlighting - Object Pascal'
```

**Remarks**
Saving to the registry must be enabled by setting the UseRegistry property to TRUE. Once this is set, any changes made by the default property editor will be automatically stored in the registry as above. TSyntaxMemoParser will not automatically read settings from the supplied RegistryKey when the component is created at application start-up. To update a parsers settings from the registry the StylesFromRegistry method must be explicitly called at application start-up.

# TSyntaxMemo REGULAR EXPRESSION SYNTAX AND USAGE

## INTRODUCTION

The demo application allows the searching and replacing of text in the editor via the Edit / Find... and Edit / Replace.. menu options. The Edit / Use Regular Expressions menu option controls when the Find... and Replace... dialogs will expect a regular expression or plain text. Plain text specification allows simple matching and replacing of exact text within the editor and should be familiar to most users. Regular expressions can be used to specify text by its characteristics rather than by the exact characters. For example, to find all identifiers in a Pascal program, it is known that the identifier can only start with certain characters and thereafter can contain only certain other characters. Regular expressions allow the specifications of such items through the use of a syntax borrowed from tools such as GREP, LEX and YACC.

The following syntax definition uses EBNF notation.

## REGULAR EXPRESSION SYNTAX

The formal description of the syntax supported by TSyntaxMemo is given as:

```
<regular expression>  ::=  [ '^' ] <expression> [ '$' ]

<expression>          ::=  <term> [ '|' <expression> ]

<term>                ::=  <factor> [ <factor> ]

<factor>              ::=  <char atom> [ <modifier> ]

<char atom>           ::=  <char>                                 |
                           <string>                               |
                           <charclass>                            |
                           '.'                                    |
                           ( '(' <expression> ')' )

<string>              ::=  '"' <charlist> '"'

<charlist>            ::=  <char> [ <charlist> ]

<charclass>           ::=  '[' [ '^' ] <CClist> ']'

<CClist>              ::=  <one CChar> [ <CClist> ]

<one CChar>           ::=  <char> [ '-' <char> ]

<modifier>            ::=  '*'                                    |
                           '+'                                    |
                           '?'                                    |

<char>                ::=  <escaped char>                         |
                           <plain char>
```

```
<escaped char>         ::=  '\' <escape sequence>

<escape sequence>     ::=  'x' <Hex value>                            |
                           '0' <Octal value>                          |
                           'n'                                        |
                           't'                                        |
                           'r'                                        |
                           'b'                                        |
                           'f'                                        |
                           's'                                        |
                           <plain char>

<Hex value>           ::=  <Hex digit> [ <Hex digit> ]

<Hex digit>           ::=  '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'
                           | '8' | '9' |
                           'a' | 'b' | 'c' | 'd' | 'e' | 'f' |
                           'A' | 'B' | 'C' | 'D' | 'E' | 'F'

<Octal value>         ::=  <Octal digit> [ <Octal digit> ] [ <Octal
                           digit> ]

<Octal digit>         ::=  '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'

<plain char>          ::=  Any visible character (See note 1)
```

**NOTES**

1. <plain char> specifications of grammar characters such as '*', '?' etc. should be escaped to prevent mis-interpretation.

2. <octal value> specifications must result in an 8 bit value. At most 3 <octal digit>s will be used to determine the <octal value>, any more will be parsed as <char atom>s after the <octal value>. If the <octal value> results in a character value > 255 (i.e. \0377) then an error will be reported. <hex value> will only accept at most 2 hex digits, any following <hex digit>s will not be used in the escape value and will be interpreted as <char atom>s. To remove ambiguity, always use a 2 digit hex code such as \x0F etc.

3. The <modifier>s are interpreted as follows:
```
   '*'   Zero or more of
   '+'   One or more of
   '?'   Zero or one of (i.e. optional)
```

4. A caret (^) at the beginning of a regular expression matches the start of a line. A dollar ($) at the end of an expression matches the end of the line. Both do NOT include the line start/end markers in the recognised text.

5. The <escaped char> letters can be upper or lower case and are interpreted as follows:
```
   \r    Carriage return(ASCII code 10)
   \n    New line       (ASCII code 13)
   \f    Form feed      (ASCII code 12)
   \t    Tab character  (ASCII code  9)
   \b    Backspace      (ASCII code  8)
   \s    Space          (ASCII code 32)
```

**REPLACEMENT TEXT SYNTAX**

Replacement text is defined by:

```
<replacement text>    ::=  <char specifier list>

<char specifier list> ::=  <char specifier> [ <char specifier list> ]

<char specifier>      ::=  <field value>
                          |
                           <escaped char>
                          |
                           <plain char>

<field value>         ::=  '%' <decimal value>

<decimal value>       ::=  <decimal digit> [ <decimal value> ]

<decimal digit>       ::=  '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'
                          | '8' | '9'
```

**NOTES**

1. <escaped char> and <plain char> are as in the regular expression syntax.
2. The <decimal value> should evaluate to a number in the range 1..10. The upper value may be changed by purchasers of the source code.

**EXAMPLES**

1. ***Locate Internet references***

   ```
   ("http://"|"mailto:"|"ftp://")[^ \n\r\"\<\\]+
   ```

   Would allow the detection of internet references that start with 'http://', 'mailto:' or 'ftp://'. The <field values> will be assigned based upon the syntax element <factor> and devolve to:

   ```
   %1.....("http://"|"mailto:"|"ftp://")
   %2.....[^ \n\r\"\<\\]+
   ```

   In english, the expression reads:

   *"Find all occurences of text that start with 'http://', 'mailto:' or 'ftp://' and are followed by at least one character that is not one of a space (\s), a newline(\n), a carriage return(\r), a quote(\"), a bracket (\<), or a slash (\\)"*

2. ***Locate all compiler directives in an Object Pascal program***

   ```
   "{$"[a-zA-Z_]+[\s\t\r\n]*[^\}]*"}"
   ```

   Would create fields as below:

   ```
   %1....."{$"
   %2.....[a-zA-Z_]+
   ```

```
%3.....[\s\t\r\n]*
%4.....[^\}]*
%5....."}"
```

In english, the expression reads:

*"Find all occurences of text that starts with a curly brace and a dollar ("{$") and is followed by at least one letter or underscore ([a-zA-Z_]+) optionally followed by some whitespace ([\s\t\r\n]*) optionally followed by any number of anything except a close curly brace ([^\}]*) and terminated by a close curly brace "}"*

## RemoveLineGlyph method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure RemoveLineGlyph(G: TGlyphIndex; L: longint);
```

**Description**
RemoveLineGlyph will ensure that a gutter glyph of index **G** is not present in line number **L**. **G** must be in the range 0..15

**Remarks**
**G** and **L** are both zero based properties.

## SaveToFile method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure SaveToFile(Filename: string);
```

**Description**
The text contents of the control will be saved to the passed filename. If the file already exists it will be overwritten.

**Remarks**
The file will be saved with CR/LF pairs terminating lines. Word-wrapping information is not saved with the text.

## SaveToStream method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
procedure SaveToStream(aStream: TStream);
```

**Description**
The text contents of the control will be saved to the passed TStream instance.

**Remarks**
The text will be written with CR/LF pairs terminating lines. Word-wrapping information is not saved with the text.

## Strings in TSyntaxMemoParser Scripts

With reference to the formal syntax of scripts, there are two types of `<string>` instances that are used in scripts. In both instances the strings are surrounded by single quotes (') and may use the following escape sequences:

| | |
|---|---|
| **\n** | The newline character = chr(13) |
| **\r** | The linefeed character = chr(10) |
| **\s** | The space character = chr(32) |
| **\t** | The TAB character = chr(9) |
| **\b** | The backspace character = chr(8) |
| **\e** | The ESCAPE character = chr(27) |
| **\xNN** | A hex character value where **N** is a hex digit `[a-fA-F0-9]` |
| **\oNNN** | An octal character value where **N** is an octal digit `[0-7]` |
| **\c** | The character **c** without any special significance attributed to **c** |
| **\<CR>** | The string is continued on the next line, if the first non-blank character on the next line is a backslash (\) then the backslash is ignored and the following character is used as the first character of the next line. No line break is inserted into the string as a result of the continuation escape sequence. |

**NB:** Escape sequence characters are not case sensitive, i.e. **\s** and **\S** both indicate the space character.

### Constant value strings

Are strings that specify exact characters that are expected, e.g. in the %%words section. Such strings follow the standard escape sequence described above and are delimited by single quotes.

### Character set specifiers

Are strings that can specify sets of characters that are expected. Within such strings (e.g. %%tokens, %%handlers etc.) the following characters have significane in addition to the above escape sequences:

| | |
|---|---|
| **[abc]** | Indicates the set composed of characters **a**, **b** and character **c**. |
| **[^abc]** | Indicates the set composed of all characters except **a**, **b** or **c**. |
| **[a-b]** | Indicates the set of characters from **a** to **b** (inclusive). Character **a** must be of equal or lower ordinal value than character **b** |

## TSyntaxMemoParser Script Reference

The **TSyntaxMemoParser** component uses a user written script to specify the format of the text to be syntax highlighted by the **TSyntaxMemo** component. Scripts are plain text files and consist of a number of sections. The formal script syntax is **provided** and should be referred to for exact specification of syntax used in each of the sections described below.

**%%language** **%%words** **%%handlers** **%%tokens** **%%effects** **%%map** **%%states** **%%containers**

### %%language section

Describes the general characteristics of the format of the text to be highlighted. Within the %%language section, the following items may be given:

| | |
|---|---|
| **Name** | A short textual description of the source text, e.g. 'Object Pascal', 'HTML' etc. |
| **Case** | Indicates whether the source text is case sensitive or not. A value of zero indicates that the source is not case sensitive, any other value indicates that the sourec should be processed with character case valid. Predefined values of **_SENSITIVE_** and **_INSENSITIVE_** should be used. |
| **Options** | The default TSyntaxMemo options used at start-up. The value specified is composed of the summation of one or more of the following values: |

| Script value | TSyntaxMemo equivalent |
|---|---|
| `__DEFAULT_OPTIONS` | `Default options` |
| `__OPT_KEYS_MOVE` | `smoKEYSMOVE` |
| `__OPT_KEYS_SELECT` | `smoKEYSSELECT` |
| `__OPT_KEYS_CLIPBOARD` | `smoKEYSCLIPBOARD` |
| `__OPT_KEYS_FUNCTIONS` | `smoKEYSFUNCTIONS` |
| `__OPT_PRINT_WRAP` | `smoPRINTWRAP` |
| `__OPT_PRINT_LINENOS` | `smoPRINTLINENOS` |
| `__OPT_PRINT_FILENAME` | `smoPRINTFILENAME` |
| `__OPT_PRINT_DATE` | `smoPRINTDATE` |
| `__OPT_PRINT_PAGENOS` | `smoPRINTPAGENOS` |
| `__OPT_WORDWRAP` | `smoWORDWRAP` |
| `__OPT_AUTOINDENT` | `smoAUTOINDENT` |
| `__OPT_TABCOLUMN` | `smoTABCOLUMN` |
| `__OPT_WRAP_OVERRIDE` | `smoWRAPOVERRIDE` |
| `__WORD_SELECT` | `smoWORD_SELECT` |

| | |
|---|---|
| **WordWrapColumn** | When the word-wrapping option of TSyntaxMemo is enabled, indicates the character column at which word-wrapping should occur. A value of zero indicates that wrapping occurs at the edge of the display window. The pre-defined value of **_EDGE** may be used to specify wrapping at the width of the window. |
| **Gutter** | Specifies the width of the gutter at the left of the text display area in pixels. The default can be specified as **_DEFAULT_GUTTER**, or any numeric value given. |
| **Anchor** | Specifies which token value should be used as a reference point when commencing analysis of the source text. This provides a known condition in the source text and if not specified, then the start of any token will be used as a reference point.<br>For sources containing such items as comments or other 'block' syntax elements, the start of these items should be specified as the anchor token so that the contents are not mis-interpreted. |

| | |
|---|---|
| **ExampleText** | A few lines of text illustrating the syntax elements of the source text. These lines will be displayed in the default property editor and it is suggested that the lines be a legal example of the source text. |
| **EditableStyles** | A list of descriptions and their token values that can be altered via the default property editor. |

## %%words section

Is used to specify syntax elements that start with a fixed set of characters. Common examples are keywords in languages (**for**, **while** etc.)
Entries in the %%words section consist of sets of specifications for each syntax 'word'. The possible entries are:

| | |
|---|---|
| **Fixed text** | Sequence of characters that must be present in the order given. Case sensitivity is applied, as specified in the %%language section above. |
| **Follow characters** | Character set specifying the characters that can legally follow the fixed text. If any other character is present then the 'word' is not recognised. |
| **Token value** | A value for the 'word' when recognised. |
| **Valid states** | A optional state set that describes in which states the 'word' can be recognised. If no state set is given then the 'word' will be recognised in the initial default state only. |

## %%handlers section

When an entry in the %%words section has been recognised it is possible to specify more precisely what can follow the fixed text. An example of this situation is comments where a comment may start with fixed characters (e.g. '/*' ) but can be followed by anything up to some other point ( e.g. '*/' ).
Entries in the %%handlers section consist of sets of specifications in the order below:

| | |
|---|---|
| **Token value** | The recognised Token value from the %%words section that is to be processed |
| **Contains specifier** | The set of characters that can follow the fixed text in the %%words section. In cases where the characters specified can optionally follow the fixed text, the character set should be specified with a question mark immediately afterwards, e.g. '[a-zA-Z]'? indicates that a letter may optionally follow the fixed text. If the contains specifier is marked as being optional then recognition will commence with the End sequence. |
| **End sequence** | Specifies what is used to recognise the end of the syntax element. Up to 3 character specifiers may be given. For example 'ing' specifies a fixed ending, '[\n\s\t\r]' specifies one whitespace character. |
| **Retain end** | Specifies whether the sequence of character(s) that were used to recognise the end of the syntax element should be considered as part of the syntax element. The pre-defined values _discard_ may be used to specify that the character(s) are not part of the syntax element, or _use_ may be used to specify that the character(s) are part of the syntax element. |

**NB**: The **Contains specifier** may be given as a pre-defined token class or the external Match(n) specifier. In these cases the **End sequence** and **Retain End** specifiers may both be omitted since the token class or Match(n) may inherently define both. The pre-defined token classes are:

| | |
|---|---|
| **_PASCAL_CHAR** | A character within a Pascal style quoted string |
| **_WEB_CHAR** | A character that forms part of an internet URL |
| **_ORIGINAL_CHAR** | An entire line of characters |
| **__STD_IDENTIFIER** | An identifier that starts with [_a-zA-Z], contains [_a-zA-Z0-9] and terminates with [^_a-zA-Z0-9] |
| **__STD_NUMBER_OR_FP** | An unsigned integer or floating point numeric constant. The number is of form <Digits> [ '.' <Digits> ] [ 'e' [ '+' \| '-' ] <Digits> ] |
| **__STD_PASCALSTRING** | A Pascal style quoted string, uses single quotes (') to delimit the string. |
| **__STD_C_STRING** | A C style quoted string, uses double quotes (") to delimit the string |
| **__STD_C_CHAR** | A C style character within a quoted string. Supports the following escape sequences: |
| **__STD_C_NUMBER_OR_FP_SIGNED** | A signed integer or floating point numeric constant of form [ '+' \| '-' ] <Digits> [ '.' <Digits> ] [ 'e' [ '+' \| '-' ] <Digits> ] |
| **__STD_NUMBER** | A decimal integer |
| **__STD_MAIL_URL** | An internet mail address |

## %%tokens section

When attempting to identify a syntax element, TSyntaxMemoParser will initially try the %%words entries (and any associated %%handlers entries). If a match is located then the syntax element has been identified and the parser will be able to update the editor. However if no match is made by the %%words entries then the %%tokens section is used to match more general syntax elements.
Entries in the %%tokens section consist of sets of specifications in the order below:

| | |
|---|---|
| **Token value** | The Token value for the syntax element described by the following specification set. |
| **Start specifier** | A single character specifier that indicates the set of characters that can start this token. This entry is required and must not overlap with any other entry in the %%tokens section. |
| **Contains specifier** | The set of characters that can follow the start character. In cases where the characters specified can optionally follow the start character, the character set should be specified with a question mark immediately afterwards, e.g. '[a-zA-Z]'? indicates that a letter may optionally follow the start character. If the contains specifier is marked as being optional then recognition will commence with the End sequence. |
| **End sequence** | Specifies what is used to recognise the end of the syntax element. Up to 3 character specifiers may be given. For example 'ing' specifies a fixed ending, '[\n\s\t\r]' specifies one whitespace character. |
| **Retain end** | Specifies whether the sequence of character(s) that were used to recognise the end of the syntax element should be considered as part of the syntax element. The pre-defined values _discard_ may be used to specify that the character(s) are not part of the syntax element, or _use_ may be used to specify that the character(s) are part of the syntax element. |
| **Valid states** | A optional state set that describes in which states the syntax element can be recognised. If no state set is given then the syntax element will be |

recognised in the initial default state only.

**NB**: The **Contains specifier** may be given as a pre-defined token class or the external Match(n) specifier. In these cases the **End sequence** and **Retain End** specifiers may both be omitted since the token class or Match(n) may inherently define both. The pre-defined token classes are as described above in the %%handlers section.

## %%effects section

Once a syntax element has been identified, TSyntaxMemo will apply syntax highlighting effects to the text of the element as per the effects described in this section.
Entries in the %%effects section consist of sets of sepecifications in the order below:

| | |
|---|---|
| **Token value** | The Token value for the syntax element described by the following specification set. |
| **Font attributes** | Specification of the Bold, Italic and Underline status of the text. The following pre-defined values should be used to indicate font effects:<br>**fsBold** Text is bold<br>**fsItalic** Text is italicised<br>**fsUnderline** Text is underlined |
| **Foreground color** | Foreground color used by the syntax element |
| **Background color** | Background color used by the syntax element |
| **Hotspot marker** | Identifies the syntax element as a 'hotspot' (See OnHyperlinkHover and OnHyperlinkClick events of TSyntaxMemo) |

**NB:** The foreground and background colors may use any 32 bit value, however the following pre-defined values may be used for convenience:
**clBlue, clAqua, clBlack, clDkGray, clFuchsia, clGray, clGreen, clLime, clLtGray, clMaroon, clNavy, clOlive, clPurple, clRed, clSilver, clTeal, clWhite, clYellow**

## %%map section

In the above %%words and %%tokens section the definition of the token value applied to each syntax element can result in a large number of possible token values. These token values can be grouped together to show all common elements as the same highlight effect through the %%map section.
By default each token value maps onto token value zero which should be defined as the default %%effect above. To override this mapping each entry in the %%map section should consist of pairs of entries thus:

| | |
|---|---|
| **Recognised token value** | The token value given in the %%words and %%tokens sections above |
| **Desired %%effect value** | Highlighting effect to be applied to the recognised token value |

Remember to override the %%effect value by mapping it to itself.

## %%states section

In the specification of source text formats and syntax, it is often the case that syntax elements should only be recognised under certain context conditions, e.g. between two syntax elements. The %%states section allows the recognition of certain syntax elements to change the set of valid states that the parser operates in. In the %%words and %%tokens sections above, it is possible to specify which states the syntax element is recognised in. By default all syntax elements are recognised in the initial start state.
Each entry in the %%states section consists of the following entries:

**Recognised token value** The token value given in the %%words and %%tokens sections above that

| | |
|---|---|
| | cause the state to change AFTER recognition. |
| **State change list** | A list of states that are turned on (+[...]) or off (-[...]) when the state starts |

Note that after a token has caused a state change the new state settings will remain until turned off by another token with an entry in the state table.

## %%container section

Provides a facility to allow syntax elements to act as containers of other syntax elements. The entries in this section are identical to the **%%states** section above although the state changes are only valid within the textual limits of the **%%container** token. Examples are quoted strings that contain relevant syntactic elements (e.g. Names, URLs etc.).

When parsing a **%%container** lexeme, the highlight effect for the %%container token will be applied to any unrecognised syntactic element located.

## Script property

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
property Script: string;
```

**Description**
Script specifies the filename of the text file containing the script for syntax element identification is located. See script reference for details of the script format.

**Remarks**
At design-time, assigning to the Script property will cause the specified script in the file to be compiled and the compiled version of the script saved with the application. At run-time assigning to the Script property will not cause an auto-compile to be performed, the CompileScript method must be explicitly invoked to compile a script at run-time.
A recompilation of a script, either at design time or run-time, will cause the previous parser model to be replaced by the new one detailed in the new script. After a successful compilation any editors using the TSyntaxMemoParser component will be updated with the new syntax effects.

## SEM_CANREDO message

Group See also Example

An application sends an SEM_CANREDO message to determine if an action can be redone by a TSyntaxMemo control.

```
wParam = <Not used>
lParam = <Not used>
```

**Parameters**

SEM_CANREDO does not take any parameters

**Return Value**

Returns **true** if an action can be redone by the SEM_REDO message, otherwise returns **false**.

## SEM_EXPORTSEL message

Group See also Example

An application sends an SEM_EXPORTSEL message to export the currently selected text of a TSyntaxMemo control to a TStream instance.

```
wParam = Formats                    // Emit format flags
lParam = TStream(aStream)           // TStream instance
```

**Parameters**

*Formats*

Is a bit field with the following bit constants of relevance:

| | |
|---|---|
| **exp_NOLF** | Export as plain text with lines delimited by a single CR (#13) |
| **exp_WITHLF** | Export as plain text with lines delimited by a CR/LF pair (#13#10) |
| **exp_RTF** | Export as Rich Text Format |

When exporting as rich text format, the exp_NOLF / exp_WITHLF settings will have no effect.

*aStream*

A TStream instance. TSyntaxMemo will export the current selection to *aStream*.Position.

**Return Value**

Does not return a value.

**Remarks**

aStream must be initialised prior to sending a SEM_EXPORTSEL message. TSyntaxMemo will export the currently selected text to the stream. The actual number of characters emitted depends upon the number of characters selected, the number of line breaks within these characters and the state of the flags of *Formats* when the message is sent.

## SEM_FINDTEXT message

Group See also Example

An application sends an SEM_FINDTEXT message to find text within a TSyntaxMemo control.

```
wParam = actions;                  // Find text control flags
lParam = PChar(pText);             // Text to be found
```

**Parameters**

*actions*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **ft_SILENT** | Locate text without displaying Find dialog |
| **ft_MATCHCASE** | Do not perform a case-insensitive search for text in pText |
| **ft_REGEXPR** | pText is a regular expression describing the text to be found |

*pText*

Points to the text used to locate the next occurence of text in the control.

**Return Value**

When **ft_SILENT** is set in actions, will return -1 for text not located, or if the text is located will return a value that is dependant upon the **ft_REGEXPR** flag in *actions*. If **ft_REGEXPR** is set in *actions* and the regular expression in *pText* matches text in the control the return value will be a TREParser instance that can be used to extract the field information from the recognised text. If **ft_REGEXPR** is not set in *actions* and *pText* is located then the return value will be the character index of the first character in the document that matches *pText*.

**Remarks**

*pText* must be initialised if **ft_SILENT** is set in *actions*, otherwise it can be **nil**.

SEM_FINDTEXT starts looking from the current insertion point. This is always the location of the flashing caret in a TSyntaxMemo. Each time a TSyntaxMemo control receives a SEM_FINDTEXT message it will re-commence looking from the current insertion point.

If **ft_SILENT** is not present in actions then the control will display the standard Windows Find dialog. Users will then be able to press the 'Find Next' button to locate the next occurence of text. **ft_SILENT** allows applications to automate the process of locating text within a control. When the Find Dialog is displayed the currently selected text will be displayed in the 'Find What' edit box. If no text is selected then the 'Find what' box will be empty.

**ft_MATCHCASE** is only used by TSyntaxMemo when **ft_REGEXPR** is not present in actions.

## SEM_GETTEXT message

Group        See also        Example

An application sends an SEM_GETTEXT message to export the currently selected text of a TSyntaxMemo control to a memory buffer.

```
wParam = UseLF;                   // Emit CR/LF pairs
lParam = PChar(pText);            // TStream instance
```

**Parameters**

*UseLF*

When TRUE, text will be exported with CR/LF pairs as line terminators. When FALSE lines will be terminated by a single CR.

*pText*

A pointer to a memory buffer to receive the text.

**Return Value**

If on entry *pText* is **nil** then will return the number of characters that will be exported (including terminating null). If *pText* is not **nil** on entry then does not return a value.

**Remarks**

*pText* must be initialised to a size of at least the number of chararacters that will be exported. TSyntaxMemo will export the currently selected text to the buffer pointed to by *pText*. The actual number of characters emitted depends upon the number of characters selected, the number of line breaks within these characters and the state of the UseLF value when the message is sent. Normally SEM_GETTEXT is called twice, once with a **nil** value for *pText* and the return value used to allocate a buffer for *pText* before SEM_GETTEXT is called again.
Use SEM_EXPORTSEL to export to a TStream instance.

## SEM_IMPORTSEL message

Group See also Example

An application sends an SEM_IMPORTSEL message to import text from a TStream instance to a TSyntaxMemo control.

```
wParam = action;                   // action to be performed
lParam = TStream(aStream);         // TStream instance
```

**Parameters**

*action*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **sKNOWNLEN** | Read in all data in *aStream*. When present TSyntaxMemo will ignore end of text stream markers (NULL or #0) and read all of the data in *aStream* until the stream is exhausted. Used when reading binary files. |
| **euDRAW** | Redraw control after new text has been imported. |
| **euNODRAW** | Do not redraw control after new text has been imported. |

*aStream*

A TStream instance. TSyntaxMemo will commence reading from the *aStream*.Position.

**Return Value**

Does not return a value.

**Remarks**

aStream must be initialised prior to sending a SEM_IMPORTSEL message. TSyntaxMemo will read the stream primarily as as a text source and will force newlines for lines in excess of 2048 characters. TSyntaxMemo will recognise embedded line breaks by the presence of a single CR, a single LF or CR followed by LF.

## SEM_INDENT message

Group See also Example

An application sends an SEM_INDENT message to indent the selected lines of a TSyntaxMemo control.

```
wParam = <Not used>
lParam = <Not used>
```

**Parameters**

Does not take any parameters.

**Return Value**

Does not return a value.

**Remarks**

The currently selected lines will be indented by inserting a space as the first character of each line. The current selection will change relative to the inserted spaces.

If WordWrap is on then lines that have been wrapped will not be indented, only the original start of the line will be indented.

## SEM_LEFTINDENT message

Group        See also        Example

An application sends an SEM_LEFTINDENT message to retrieve or set the current start column of the left edge of the display in a TSyntaxMemo control.

```
wParam = action;         // action to be performed
lParam = leftedge;       // left edge value
```

**Parameters**

*action*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **eaSET** | Set the new left edge column number leftedge . The **euDRAW** / **euNODRAW** bits determine whether the control is repainted after the new left edge value is set. |
| **eaGET** | Return the current left edge value as a result |
| **euDRAW** | Only applicable for **eaSET**. Redraw control after new left edge is set. |
| **euNODRAW** | Only applicable for **eaSET**. Do not redraw control after new left edge is set. |

*leftedge*

For **eaSET**, is the new value of the left edge of the control. For **eaGET** leftedge is not used. *leftedge* is a zero based column number.

**Return Value**

For *action* = **eaGET** returns the 32 bit, zero based left edge column number of the current display state . For *action* = **eaSET** the return value is undefined.

**Remarks**

If setting a new left edge value (*action* = **eaSET**), then the new left edge value may result in no text being displayed if the lines in view do not extend to the new left edge value.

## SEM_MODIFIED message

Group See also Example

An application sends an SEM_MODIFIED message to get or set the modified state of a TSyntaxMemo control.

```
wParam = action;                  // action to be performed
lParam = Boolean(nModified);      // new Modified state values
```

**Parameters**

*action*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **eaSET** | Set the new option flags to value of nOptions in lParam. |
| **eaGET** | Return the current options flags as a result |

*nModified*

When **eaSET** is set in action then nModified is the new state of the Modified property of the TSyntaxMemo control.

**Return Value**

When **eaGET** is set in action the return value is TRUE if the contents of the control have been modified, otherwise it is FALSE.

**Remarks**

TSyntaxMemo normally takes care of the modified state of a control by tracking the undo / redo actions. As a default action TSyntaxMemo will reset the Modified property if text is entered into the control via the LoadFromFile, LoadFromStream or via the Windows WM_SETTEXT message. Applications may process saving the text on their own and should send a SEM_MODIFIED message to reset the state of the control. If the effect is to reset the Modified state then TSyntaxMemo will dump all saved undo / redo actions.
The TSyntaxMemo.Modified public property is translated to SEM_MODIFIED messages and acts in the same manner as described here.

## SEM_OPTIONS message

Group    See also    Example

An application sends an SEM_OPTIONS message to retrieve or set the current options of a TSyntaxMemo control.

```
wParam = action;                  // action to be performed
lParam = nOptions;                // new options values
```

**Parameters**

*action*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **eaSET** | Set the new option flags to value of nOptions in lParam. |
| **eaGET** | Return the current options flags as a result |
| **euDRAW** | Only applicable for **eaSET**. Redraw control after new options are installed. Control will auto scroll to ensure the end of the new selection is in view. |
| **euNODRAW** | Only applicable for **eaSET**. Do not redraw control after new options are installed. |

*nOptions*

A 32 bit value that is interpreted as a bit field with the following bit masks applicable (default states indicated):

| | |
|---|---|
| **eo_KEYS_MOVE** | Use default movement   keys (ON) |
| **eo_KEYS_SELECT** | Use default selection keys (ON) |
| **eo_KEYS_CLIPBOARD** | Use default clipboard keys (ON) |
| **eo_KEYS_FUNCTIONS** | Use default functions (ON) |
| **eo_PRINT_WRAP** | Wrap lines when printing (ON) |
| **eo_PRINT_LINENOS** | Print line numbers (ON) |
| **eo_PRINT_FILENAME** | Print filename in header (ON) |
| **eo_PRINT_DATE** | Print date in header (ON) |
| **eo_PRINT_PAGENOS** | Print page numbers (ON) |
| **eo_WORDWRAP** | Word-wrap lines at right margin (OFF) |
| **eo_DEFAULT** | Default options (as above) |

**Return Value**

For **eaGET** set in *actions*, returns the current options flags, otherwise does not return a value.

**Remarks**

The key assignments affected by the **eo_KEYS_xxx** flags are given in the **keyboard interface** section.

**eo_TELL_ERRORS** when present will cause a dialog to be displayed should TSyntaxMemo detect any form of error. If the flag is not set then TSyntaxMemo will raise an exception for errors it detects allowing applications to deal with errors.

Printing flags are detailed in the Print method topic.

## SEM_REDO message

Group See also Example

An application sends an SEM_REDO message to re-do an action in a TSyntaxMemo control.

```
wParam = <Not used>
lParam = <Not used>
```

**Parameters**

SEM_REDO does not take any parameters

**Return Value**

Does not return a value

**Remarks**

If there is no action that can be re-done when the SEM_REDO message is received by a TSyntaxMemo control then no action will be performed. Use SEM_CANREDO to determine if there is at least one re-do action available.

## SEM_REPARSE message

Group See also Example

An application sends an SEM_REPARSE message to update the display style of a TSyntaxMemo control from its current TSyntaxMemoParser control.

```
wParam = <Not used>
lParam = <Not used>
```

**Parameters**

SEM_REPARSE does not take any parameters

**Return Value**

SEM_REPARSE does not return a result.

**Remarks**

SEM_REPARSE is sent to a **TSyntaxMemo** control to refresh its display of text. The entire contents are re-scanned and the settings of the active **TSyntaxMemoParser** component are used to both parse the text and display syntax highlighting. When the settings of a **TSyntaxMemoParser** control are changed, the attached editors can be refreshed either by invoking the **UpdateEditors** method or by sending a SEM_REPARSE message to each of the **TSyntaxMemo** controls using the **TSyntaxMemoParser** component. Use of **UpdateEditors** is the preferred method.

## SEM_REPLACESEL message

Group See also Example

An application sends an SEM_REPLACESEL message to replace the current selection of text in a TSyntaxMemo control by new text in a memory buffer.

```
wParam = action;          // action to be performed
lParam = PChar(pText);    // text to be inserted
```

**Parameters**

*action*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **euDRAW** | Redraw control after new text has been imported. |
| **euNODRAW** | Do not redraw control after new text has been imported. |

*pText*

A PChar instance that is used as the source of the text to replace the currently selected text.

**Return Value**

Does not return a value.

**Remarks**

*pText* can be **nil** on entry, in which case the current selection will be deleted. TSyntaxMemo will read the text pointed to by pText until a NULL is encountered. TSyntaxMemo will read the stream primarily as as a text source and will force newlines for lines in excess of 2048 characters. TSyntaxMemo will recognise embedded line breaks by the presence of a single CR, a single LF or CR followed by LF.

## SEM_REPLACETEXT message

Group See also Example

An application sends an SEM_REPLACETEXT message to replace text within a TSyntaxMemo control.

```
wParam = actions;                 // Find text control flags
lParam = <Not used>
```

**Parameters**

*actions*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **ft_MATCHCASE** | Do not perform a case-insensitive search for text in pText |
| **ft_REGEXPR** | pText is a regular expression describing the text to be found |

**Return Value**

Does not return a value.

**Remarks**

SEM_REPLACETEXT starts looking from the current insertion point. This is always the location of the flashing caret in a TSyntaxMemo. Each time a TSyntaxMemo control receives a SEM_REPLACETEXT message it will re-commence looking from the current insertion point.

**ft_MATCHCASE** is only used by TSyntaxMemo when **ft_REGEXPR** is not present in actions.

## SEM_SELECTION message

Group See also Example

An application sends an SEM_SELECTION message to retrieve or set the current selection of text in a TSyntaxMemo control.

```
wParam = action;                   // action to be performed
lParam = pChRange(range);          // pointer to range of characters structure
```

**Parameters**

*action*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **eaSET** | Set the new selection from values pointed to by *range*. The **euDRAW** / **euNODRAW** bits determine whether the control is repainted after the new selection is set. |
| **eaGET** | Return the current selection in the structure pointed to by *range* |
| **euDRAW** | Only applicable for **eaSET**. Redraw control after new selection is set. Control will auto scroll to ensure the end of the new selection is in view. |
| **euNODRAW** | Only applicable for **eaSET**. Do not redraw control after new selection is set. |

*range*

Points to a **TChRange** structure used to either receive the current selection (*action* = **eaGET**) or to set the new selection (*action* = **eaSET**).

**Return Value**

Does not return a value.

**Remarks**

If setting a new selection (*action* = **eaSET**), then the new selection range will be restricted to the extent of text in the control. To select all text in the control set the chStart field of range to -1.

## SEM_TOPLINEINDEX message

Group    See also    Example

An application sends an SEM_TOPLINEINDEX message to retrieve or set the current top line of the display in a TSyntaxMemo control.

```
wParam = action;         // action to be performed
lParam = topline;        // pointer to range of characters structure
```

**Parameters**

*action*

Value of wParam. Specifies the actions to be performed. Uses bit fields as below:

| | |
|---|---|
| **eaSET** | Set the new top line of the control to the value in topline . The **euDRAW** / **euNODRAW** bits determine whether the control is repainted after the new topline is set. |
| **eaGET** | Return the current top line index as a result |
| **euDRAW** | Only applicable for **eaSET**. Redraw control after new top line is set. |
| **euNODRAW** | Only applicable for **eaSET**. Do not redraw control after new top line is set. |

*topline*

For **eaSET**, is the new value of the top line of the control. For **eaGET** topline is not used. *topline* is a zero based line number.

**Return Value**

For *action* = **eaGET** returns the 32 bit, zero based line number of the current line at the top of the display. For *action* = **eaSET** the return value is undefined.

**Remarks**

If setting a new top line value (*action* = **eaSET**), then the new top line value will be restricted extent of text in the control.

## SEM_UNDENT message

Group        See also        Example

An application sends an SEM_UNDENT message to un-indent the selected lines of a TSyntaxMemo control.

```
wParam = <Not used>
lParam = <Not used>
```

**Parameters**

Does not take any parameters.

**Return Value**

Does not return a value.

**Remarks**

The currently selected lines will be un-indented by deleting any space at the start of each line in the selection. If a line does not have a space as its first character then it will be unaffected. The current selection will change relative to the deleted spaces.

If WordWrap is on then lines that have been wrapped will not be unindented, only the original start of the line will be unindented.

IsBookmarkSet
ClearBookmark

## SetBookmark method

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`procedure SetBookmark(n: byte; Line, Col: longint);`

**Description**
Sets a bookmark number from zero to nine (**n**) to the specified Line (**Line**) and Column (**Col**) within the text of the control. If **Line** is beyond the last line of text then no action will be taken. If **Col** is beyond the last character on the line then the bookmark will be set to the last character on the line (but will remember the desired column should the line extend later).

**Remarks**

## StylesAsString method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
function StylesAsString: string;
```

**Description**
StylesAsString retrieves the current settings of the control and returns them as a formatted string. The items stored within the formatting of the string are:

- **Options,**
- **Gutter width,**
- **Word-wrap column,**
- **Indent step amount,**
- **Word-wrap override marks,**
- **%%effects settings**

The string value returned can be stored as desired. To set the styles from a formatted string, as returned by the StylesAsString method of the same parser, using the same script, use the StylesFromString method.

**Remarks**

## StylesFromRegistry method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
procedure StylesFromRegistry(UseDefault: Boolean; aKey: string);
```

**Description**
StylesFromRegistry allows the restoration of the parser and TSyntaxMemo settings defined in a registry entry. If **UseDefault** is TRUE then the RegistryKey value will be used as the registry base, otherwise the **aKey** value will be used as the registry base from which the entries should be restored.
The exact registry key from which settings will be restored is detailed in RegistryKey.

**Remarks**
See StylesAsString for details of the settings that can be restored from the registry using StylesFromRegistry.

## StylesFromString method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
procedure StylesFromString(Styles: string);
```

**Description**
StylesFromString allows the restoration of the parser and TSyntaxMemo settings defined in a formatted string. The string must have been returned as the result of the StylesAsString method or must have been saved in the registry by the component.

**Remarks**
See StylesAsString for details of the settings that can be restored from the registry using StylesFromString.

## StylesToRegistry method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
procedure StylesToRegistry;
```

**Description**
StylesToRegistry will force the component to save its current settings using the RegistryKey property. UseRegistry MUST be true before StylesToRegistry is invoked otherwise an exception will be raised.

**Remarks**

## TCharTest type

**Unit**
SyntaxEd

**Declaration**
```
TCharTest = function(c: char): Boolean of object;
```

**Description**
TSyntaxMemo will generate TCharTest events whenever it is required to test a character. fn_CharIsInWord and fn_CharStartsWord are two events that TSyntaxMemo generates to allow applications to override the definition of characters that make up 'words'. TCharTest event handlers will be passed a single character to test and should return true if the condition is met, false otherwise.

**Remarks**
At present TSyntaxMemo will only generate events when selecting words (double click), moving to word boundaries (**Ctrl+Left Arrow** / **Ctrl+Right Arrow**). TSyntaxMemo may be upgraded in the future to allow further processing of words within files. These upgrades will use any attached event handlers to carry out the required checks.

## TChRange type

A range of characters within a **TSyntaxMemo** document

```
TChRange = record
             chStart,
             chEnd      : longint;
           end;
```

**TChRange** is used by many of the API messages to specify which characters are to be affected during an operation.

## TFormatEntry type

**Declaration**

```
TFormatEntry = record
   fme_style:        TFontStyles;
   fme_color:        TColor;
   fme_background:   TColor;
   fme_IsHotspot:    Boolean;
   fme_FontValid:    longint;
 end;
```

**Description**

TFormatEffect describes the syntax highlighting applied to individual syntax token values. The fields are used as below:

| | |
|---|---|
| `fme_style` | Bold, Italic, Underline style of text |
| `fme_color` | Color of text |
| `fme_background` | Background color of text |
| `fme_IsHotspot` | = true if syntax token is a hotspot |
| `fme_FontValid` | <> 0 if effect is valid |

## TGlyphIndex type

**Declaration**
```
TGlyphIndex = word;
```

**Description**
TGlyphIndex is used as a bit field to describe the glyphs that are currently displayed for a line within TSyntaxMemo. At present up to 16 different glyphs may be used. If a bit is set in a TGlyphIndex value then it indicates that the corresponding ordinal glyph image in the GutterGlyphs property will be displayed.

## TREParser class

See also

**Unit**
ReClass

**Declaration**

```
TREParser = class
    public
      property  Sections[I: longint]: longint      read;
      property  NumSections          : longint      read;
    end;
```

**Description**
TREParser is a general purpsose regular expression parser and recognition engine. It is used by TSyntaxMemo to deal with regular expression specifications for text to be located in the Find / Replace functions of TSyntaxMemo.

TSyntaxMemo deals with the SEM_FINDTEXT and SEM_REPLACETEXT messages that specify a regular expression as the source and return an instance of TREParser. The returned value can be cast to an instance of **TREParser** and the above public properties examined for recognised text extents.

**Public properties**

```
Sections[I: longint]: longint      read;
```

A read only property that holds character indexes of the start of each **<factor>** of the regular expression.

```
NumSections          : longint      read;
```

A read only property that indicates the maximum number of sections contained in the regular expression last used by TREParser.

Other properties and methods of TREParser are not documented.

## Installing TSyntaxMemo

Extract the files in the supplied ZIP file to a new directory. Then:

**Delphi 2**

1. Select **Install component** from the main menu **component** menu item
2. Select SyntaxEd.pas in the directory created above
3. Press OK

**Delphi 3**

1. Select **Install component** from the main menu **component** menu item
2. In the dialog that appears select the **Into new package** tab
3. For the **Unit file name**, select SyntaxEd.pas in the directory created above
4. For the package name, enter **dbSoft**
5. Enter a description of the package
6. Press OK

TSyntaxMemo and TSyntaxMemoParser will now be installed in the **dbSoft** component tab

## Installing the help file

**Delphi 2**

1. Copy the **TSynMemo.hlp** and **TSynMemo.kwf** into the **HELP** directory of your Delphi 2 installation
2. Run **HelpInst** from the **\HELP** directory and open the **Delphi.kwf** file in your **\BIN** directory
3. Click the '+' button and open the **TSynMemo.kwf** as copied in step 1
4. Click the Save button to re-generate the **Delphi.kwf** keyword file

**Delphi 3**

1. In the **\HELP** directory of the Delphi 3 installation edit the **Delphi3.cfg** file and add the following line at the end:
   ```
   :link TSynmemo.hlp
   ```
2. Save the modified **Delphi3.cfg**
3. Delete the **Delphi3.GID** file in the **\HELP** directory
4. When Delphi 3 is started press F1 to re-build the on-line help. After this normal F1 style help will be available for the **TSyntaxMemo** and **TSyntaxMemoParser** components, their methods, properties and other types / constants.

## TSyntaxMemo screenshot

Image not available

## TSyntaxMemo

See also		Properties		Methods		Events		Tasks

TSyntaxMemo is a multiline edit control that can display and edit syntax highlighted text.

**Unit**
SyntaxEd

**Description**
TSyntaxMemo is a syntax highlighting editor component. TSyntaxMemo implements all properties, methods, events and messages of TMemo and can be used as a direct replacement for TMemo. TSyntaxMemo interfaces with TSyntaxMemoParser components to provide the analysis phase of syntax highlighting. Up to 6 different TSyntaxMemoParser components may be used by each TSyntaxMemo conmponent, each TSyntaxMemoParser component defining a different source code format to be highlighted. The selected TSyntaxMemoParser component is controlled through the ActiveParser property of TSyntaxMemo.

In addition to the functionality of TMemo, TSyntaxMemo also provides facilities as below:
  Unlimited undo/redo of editing actions
  Gutter area for display of user defined glyphs
  Bookmark support
  Regular expression searching and replacing

**Remarks**

# TDBSyntaxMemo

See also Properties Methods Events Tasks

TDBSyntaxMemo is a multiline edit control that can display and edit a syntax highlighted field in a dataset.

**Unit**
SyntaxEd

**Description**
Use TDBSyntaxMemo to let users edit a field that may contain lengthy textual data or to simply display the contents of such a field with syntax highlighting applied to the contents. TDBSyntaxMemo uses the Text property to represent the contents of the field.

TDBSyntaxMemo permits multiple lines of text. Thus, TDBSyntaxMemo is appropriate for long alphanumeric fields or text binary large objects (BLOBs). For short alphanumeric fields, consider using a TDBEdit component instead.

If the application doesnt require the data-aware capabilities of TDBSyntaxMemo, use a memo control (TMemo or TSyntaxMemo) instead, to conserve system resources.

## TSyntaxMemoParser - formal script syntax definition

Uses EBNF syntax specification format

```
<Script>                    ::=    [ <Language section> ]
                                   [ <Word section> ]
                                   [ <Handlers section> ]
                                   [ <Tokens section> ]
                                   [ <Effects section> ]
                                   [ <Map section> ]
                                   [ <States section> ]
                                   [ <Containers section> ]

<Language section>          ::=    '%%language' [ <Language item list> ]
<Language item list>        ::=    <One language item> [ <Language item list> ]
<One language item>         ::=    <Name item>
                                   |    <Case item>
                                   |    <Options item>
                                   |    <WordWrapColumn item>
                                   |    <Anchor item>
                                   |    <Gutter item>
                                   |    <ExampleText item>
                                   |    <EditableStyles item>
                                   |    <Wrapoverride item>

<Name item>                 ::=    'Name' '=' <string>
<Options item>              ::=    'Options' '=' <Expression>
<WordWrapColumn item>       ::=    'WordWrapColumn' '=' <Expression>
<Anchor item>               ::=    'Anchor' '=' <ByteExpression>
<Gutter item>               ::=    'Gutter' '=' <Expression>
<ExampleText item>          ::=    'ExampleText' '=' <string>
<EditableStyles item>       ::=    <One editable style> [ ',' <EditableStyles
                                   item> ]
<One editable style>        ::=    '(' <string> ',' <Expression> ')'
<Wrapoverride item>         ::=    'WrapOverride' '=' <string>

<Word section>              ::=    '%%words' [ <Word list> ]
<Word list>                 ::=    <One word> [ <Word list> ]
<One word>                  ::=    <string> <EndcharStyle> <ByteExpression>
                                   [ <State specifier> ]
<EndCharStyle>              ::=    <string>
<State specifier>           ::=    '[' <Byte Expression list> ']'
<Expression list>           ::=    <ByteExpression> [ <ByteExpression list> ]

<Handlers section>          ::=    '%%handlers' [ <Handler list> ]
<Handler list>              ::=    <One handler> [ <Handler list> ]
<One handler>               ::=    <ByteExpression> <Remainder specifier>
<Remainder specifier>       ::=    <Contains specifier>  [ <Ends specifier>
                                   <Ends action> ]
<Contains specifier>        ::=    ( <Token function ID> | <string> ) [ '?' ]
<Ends specifier>            ::=    <string>
<Ends action>               ::=    <Expression>

<Tokens section>            ::=    '%%tokens' [ <Token list> ]
<Token list>                ::=    <One token> [ <Token list> ]
<One token>                 ::=    <ByteExpression> ( <Token Function> | <Token
                                   specifier> ) [ <State specifier> ]
<Token function>          ::=    <Expression>
<Token specifier>         ::=    <Start set sepecifier> <Contains specifier>
                                 <Ends specifier> <Ends action>

<Effects section>         ::=    '%%effects' [ <Effects list> ]
<Effects list>            ::=    <One effect> [ <Effects list> ]
<One effect>              ::=    <ByteExpression> <Font style> <Text color>
                                 <Background color> <Hotspot specifier>
<Font Style>              ::=    '[' [ <Font style list> ] ']'
<Font style list>         ::=    <Font Expression list>
<Expression list          ::=    <Font Expression> [ <Font Expression list> ]
<Text color>              ::=    <Expression>
<Background color>        ::=    <Expression>
<Hotspot specifier>       ::=    [ 'hotspot' ]

<Map section>             ::=    '%%map' [ <Map list> ]
<Map list>                ::=    <One map item> [ <Map list> ]
<One map item>            ::=    <Script token value> <Applied affect value>
<Script token value>      ::=    <ByteExpression>
<Applied effect value>    ::=    <ByteExpression>

<States section>          ::=    '%%states' [ <State list> ]
<State list>              ::=    <One state transition> [ <State list> ]
<One state transition>    ::=    <ByteExpression> '(' [ <State changes> ] ')'
<State changes>           ::=    ( '+' | '-' ) <State specifier>

<Containers section>      ::=    '%%containers' [ <State list> ]

<Expression>              ::=    <Factor> [ <Add op> <Expression> ]
<Factor>                  ::=    <Integer>
                                 | '(' <Expression> ')'
<Add op>                  ::=    '+' | '-'
```

**NOTES**: `<ByteExpression>` evaluates to the range 0..255
`<Font expression>` should evaluates to one of:
ord(fsBold)
ord(fsUnderline)
ord(fsItalic)
Any other value may have undesired side-effects.

See string syntax for details of character sets, constant strings and escape sequences.

## TSyntaxMemoParser tasks index

## TSyntaxMemoParser

See also Properties Methods Events Tasks

**Description**
A non-visual VCL component that provides parsing functionality to TSyntaxMemo editor components. TSyntaxMemoParser uses a script file to specify the method used in parsing the text contents of TSyntaxMemo. The script syntax is described in detail in the Tasks section.
TSyntaxMemoParser can be used with more than one TSyntaxMemo control, in this manner, each TSyntaxMemo control that uses the same TSyntaxMemoParser will display its text using the same highlighting effects. Changing the highlighting effects of one TSyntaxMemo control at run-time or design time will cause all other TSyntaxMemo controls using the same TSyntaxMemoParser to be automatically updated with the new effect styles.

Although TSyntaxMemoParser uses a text file script to specify the parsing methods, this script is compiled by the control at design time and the compiled format is saved with the application. There is no need to supply a script with applications that use TSyntaxMemoParser controls since TSyntaxMemoParser will read the compiled data if the original script cannot be found.

Run-time modification of the parsing methods can be achieved by forcing the TSyntaxMemoParser to compile another script via the CompileScript or CompileFromStream methods. Both cause the associated TSyntaxMemo control(s) to be updated with the changes.

**Remarks**
Script creation is covered in the tasks section of this help file. However pre-written scripts can be downloaded from the TSyntaxMemo Web site. The author will provide full support and advice on the creation of scripts.

## UpdateEditors method

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
procedure UpdateEditors;
```

**Description**
UpdateEditor is used to reflect changes to the parser settings in attached TSyntaxMemo editors. Invoking the UpdateEditors method will cause attached editor components to be re-drawn using the current TSyntaxMemoParser settings.

**Remarks**

## UseRegistry property

See also Example

**Applies to**
TSyntaxMemoParser controls

**Declaration**
```
property UseRegistry: Boolean;
```

**Description**
UseRegistry controls the behaviour of the components interface with the Windows Registry via the RegistryKey property.

**Remarks**
See RegistryKey for full details of the Windows Registry interface.

## WrapAtColumn property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
```
property WrapAtColumn: word;
```

**Description**
In word-wrap mode the text in the control will be wrapped at the WrapAtColumn character column.

**Remarks**
Setting WrapAtColumn to zero will cause the text to be wrapped at the width of the display area. The WrapOverride property allows word-wrapping to be overriden for certain lines within the control.

## WrapOverride property

See also Example

**Applies to**
TSyntaxMemo controls

**Declaration**
`property WrapOverride: string;`

**Description**
WrapOverride specifies the start character of lines that will not be word-wrapped. Each character in the property indicates a possible line start character.

**Remarks**
In order for WrapOverride to be operational, Options must contain the **smoWrapOverride** and **smoWordWrap** flags. WrapOverride defaults to the empty string (i.e. no lines are overriden in word-wrap mode).