TMS HTML Controls Pack
DEVELOPERS GUIDE

Jul 2016

Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability

The HTML enabled components are available for Delphi 7, 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle, C++Builder 2007, 2009, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, 10 Seattle (Prof/Enterprise/Architect)

The HTML enabled components have been designed for and tested with: Windows Vista, Windows 7, Windows 8, Windows 10.

Online references

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

TMS HTML Controls Pack page:
http://www.tmssoftware.com/site/htmlcontrols.asp

TMS mini HTML formatting tags supported:
http://www.tmssoftware.com/site/minihtml.asp

HTLM Controls and tag reference

TMS HTML Controls comprise a range of controls that have the capability to display tekst decorated with HTML tags to draw better attention to what the application wants to bring in focus, add images etc..
TMS HTML Controls are based on a compact & fast HTML rendering engine that supports a subset of HTML tags, referred to as mini HTML.

This is a list of tags supported in our range of HTML controls:

B : Bold tag
<B> : start bold text
</B> : end bold text

Example : This is a <B>test</B>

U : Underline tag
<U> : start underlined text
</U> : end underlined text

Example : This is a <U>test</U>

I : Italic tag
<I> : start italic text
</I> : end italic text

Example : This is a <I>test</I>

S : Strikeout tag
<S> : start strike-through text
</S> : end strike-through text

Example : This is a <S>test</S>

A : anchor tag
<A href="value"> : text after tag is an anchor. The 'value' after the href identifier is the anchor. This can be an URL (with ftp,http,mailto,file identifier) or any text.
If the value is an URL, the shellexecute function is called, otherwise, the anchor value can be found in the OnAnchorClick event </A> : end of anchor

Examples : This is a <A href= "mailto:myemail@mail.com ">test</A>
This is a <A href="http://www.tmssoftware.com">test</A>
This is a <A href="somevalue">test</A>

FONT : font specifier tag
<FONT face='facevalue' size='sizevalue' color='colorvalue' bgcolor='colorvalue'> : specifies font of text after tag.
with
face : name of the font
size : HTML style size if smaller than 5, otherwise pointsize of the font
color : font color with either hexidecimal color specification or Borland style color name, ie clRed,clYellow,clWhite ... etc
bgcolor : background color with either hexidecimal color specification or Borland style color name
</FONT> : ends font setting

Examples : This is a <FONT face="Arial" size="12" color="clred">test</FONT>
This is a <FONT face="Arial" size="12" color="#FF0000">test</FONT>

P : paragraph
<P align="alignvalue" [bgcolor="colorvalue"] [bgcolorto="colorvalue"]> : starts a new paragraph, with left, right or center alignment. The paragraph background color is set by the optional bgcolor parameter. If bgcolor and bgcolorto are specified,
a gradient is displayed ranging from begin to end color.
</P> : end of paragraph

Example : <P align="right">This is a test</P>
Example : <P align="center">This is a test</P>
Example : <P align="left" bgcolor="#ff0000">This has a red background</P>
Example : <P align="right" bgcolor="clYellow">This has a yellow background</P>
Example : <P align="right" bgcolor="clYellow" bgcolorto="clred">This has a gradient background</P>*

HR : horizontal line
<HR> : inserts linebreak with horizontal line. The default color is black. To set the color of the horizontal line, the COLOR attribute can be used:
<HR COLOR="clBlack">

BR : linebreak

 : inserts a linebreak

BODY : body color / background specifier
<BODY bgcolor="colorvalue" [bgcolorto="colorvalue"] [dir="v|h"] background="imagefile specifier"> : sets the background color of the HTML text or the background bitmap file

Example : <BODY bgcolor="clYellow"> : sets background color to yellow
<BODY background="file://c:\test.bmp"> : sets tiled background to file test.bmp
<BODY bgcolor="clYellow" bgcolorto="clWhite" dir="v"> : sets a vertical gradient from yellow to white

IND : indent tag
This is not part of the standard HTML tags but can be used to easily create multicolumn text
<IND x="indent"> : indents with "indent" pixels

Example :
This will be <IND x="75">indented 75 pixels.

IMG : image tag
<IMG src="specifier:name" [align="specifier"] [width="width"] [height="height"] [alt="specifier:name"] > : inserts an image at the location

specifier can be : idx : name is the index of the image in the associated imagelist
ssys : name is the index of the small image in the system imagelist or a filename for which the corresponding system imagelist is searched
lsys : same as ssys, but for large system imagelist image
file : name is the full filename specifier
res : name of a resource image (not visible at design time) Supported image types: BMP, GIF, PNG. Please note that the resource type should be RT_RCDATA!
no specifier : name of image in an PictureContainer

Optionally, an alignment tag can be included. If no alignment is included, the text alignment with respect to the image is bottom. Other possibilities are : align="top" and align="middle"

The width & height to render the image can be specified as well. If the image is embedded in anchor tags, a different image can be displayed when the mouse is in the image area through the Alt attribute.

Examples : This is an image <IMG src="idx:1" align="top">
This is an image <IMG src="ssys:1"> and another one <IMG src="ssys:worfile.doc">
This is an image <IMG src="file://c:\my documents\test.bmp">
This is an image <IMG src="res://BITMAP1">
This is an image <IMG src="name">

SUB : subscript tag
<SUB> : start subscript text
</SUB> : end subscript text

Example : This is ⁹/₁₆ looks like 9/16

SUP : superscript tag
<SUP> : start superscript text
</SUP> : end superscript text

BLINK : blink tag(supported in TAdvStringGrid and descendants and THTMListbox)
<BLINK> : start blinking text
</BLINK> : stop blinking text

Example : This is <FONT color="clred"><BLINK>blinking red</BLINK></FONT>text.

UL : list tag
<UL> : start unordered list tag
</UL> : end unordered list

Example : <UL>
<LI>List item 1
<LI>List item 2
<UL>
<LI> Sub list item A
<LI> Sub list item B
</UL>
<LI>List item 3
</UL>

LI : list item
<LI [type="specifier"] [color="color"] [name="imagename"]> : new list item
specifier can be "square" or "circle" or "image" bullet color sets the color of the square or circle bullet imagename sets the PictureContainer image name for image to use as bullet

SHAD : text with shadow
<SHAD> : start text with shadow
</SHAD> : end text with shadow

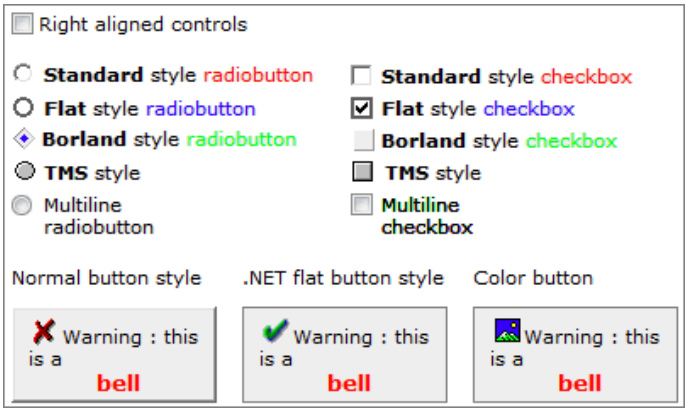
Z : hidden text
<Z> : start hidden text
</Z> : end hidden text

HI : hilight
<HI> : start text hilighting
</HI> : stop text hilighting

E : Error marking
<E> : start error marker
</E> : stop error marker

Special characters
Following standard HTML special characters are supported :
< : less than : <
> : greater than : >
& : &
" : "
  : non breaking space
™ : trademark symbol
€ : euro symbol
§ : section symbol
© : copyright symbol
¶ : paragraph symbol

THTLMButtons



THTMLButtons description

Standard, flat, Borland and TMS Checkbox, Radiobutton, RadioGroup, Button with HTML formatted captions.

THTMLButtons features

Standard, flat, Borland and TMS style Checkbox and RadioButton control with multiline & HTML formatted caption capability.

- THTMLCheckBox
- THTMLRadioButton
- THTMLRadioGroup
- THTMLCheckGroup
- THTMLButton
- TDBHTMLCheckBox
- TDBHTMLRadioGroup

THTMLButtons use

The HTML enabled radiobutton, checkbox, radiogroup and checkgroup are very similar to the standard VCL radiobutton, checkbox, radiogroup. The THTMLCheckGroup is similar to a radiogroup control but contains checkboxes. The differences between a regular VCL radiobutton, checkbox, radiogroup and the TMS equivalent controls are:

Selection of button styles:

Following button styles can be selected with the ButtonType property:

btBorland	Borland style (chiseled look) radiobutton/checkbox	THTMLRadioButton THTMLCheckBox
btClassic	Classic Windows 2000 style radiobutton/checkbox	THTMLRadioButton THTMLCheckBox
btFlat	Flat Windows 2000 style radiobutton/checkbox	THTMLRadioButton THTMLCheckBox

btTheme	Automatically adapts radiobutton/checkbox to operating system theme	
btTMS	Gray 3D TMS style radiobutton/checkbox	
btWinXP	Fixed XP themed radiobutto/checkbox irrespective of operating system	

Button vertical alignment:

As the checkbox or radiobutton can have a wordwrapped caption, the vertical alignment of the radiobutton/checkbox wrt the caption can be set with the ButtonVertAlign property.

Checkmark color:

When the button type is set to btBorland, the color of the checkmark is set with the CheckColor property.

Ellipsis text drawing:

When the text is too long to fit in the control, the text is displayed with an ellipsis (…) at the end to indicate the text is longer than what can be displayed;

HTML formatted / wordwrapped / transparent text:

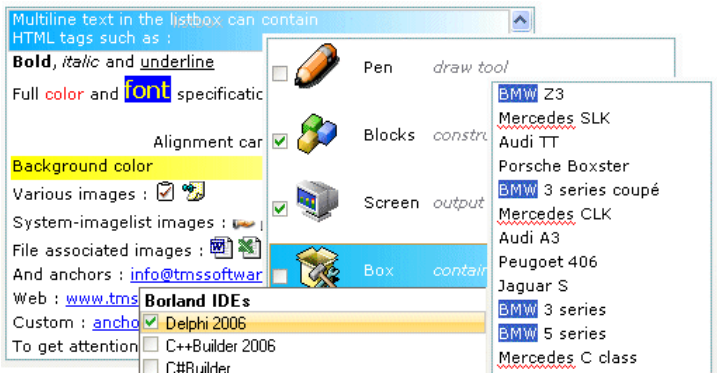
The caption of the checkboxes and radiobuttons can have HTML formatted text. Please refer to the chapter about the TMS mini HTML engine for all tags supported. Via the PictureContainer and ImageList, images can be set with the <IMG> tag. The HTML formatted text can automatically wordwrap and when the property Transparent is set true, the control is drawn with true transparency on the parent control.

The THTMLCheckGroup has no equivalent VCL control. The THTMLCheckGroup is a group of THTMLCheckBoxes. This means that for each checkbox, the HTML formatted caption text can be set via HTMLCheckGroup.Items[index: integer]: string.

A few additional options are available:

To get and set the checkbox state for each checkbox in the group, the property HTMLCheckGroup.Checked[index: integer]: Boolean can be used. Each checkbox can also be set separately as readonly with HTMLCheckGroup.ReadOnly[index: integer]: Boolean. Finally, each checkbox can also dynamically be set as enabled or disabled with the event OnIsEnabled. This event is triggered whenever the checkbox group is updated.

THTMListbox & THTMLCheckList



THTMListbox & THTMLCheckList description

A listbox & checklist that can contain text with HTML formatting capabilities.

THTMListbox & THTMLCheckList features

- A listbox with HTML formatting capabilities
- Supports a subset of the HTML formatting tags
- Standard, Office 2003, Office 2007 selection colors
- HTML aware lookup and sorting and access of text without HTML tags
- Supports the PictureContainer for adding GIF, JPEG, BMP, ICO or WMF files to the listbox
- Supports animated GIFs
- Direct text highlighting in listbox as well as error marking
- Capability to have item hints, anchor hints
- Capability to have blinking text
- Capability to have comment (non checkbox) items in HTMLCheckList
- Compatible with TAdvFormStyler, TAdvAppStyler
- and much more ...

THTMListbox & THTMLCheckList use

The THTMListbox and THTMLCheckList are basically equivalent components for the standard VCL TListbox and TCheckList where each item in the list supports HTML formatted text. For full details about all HTML supported tags in the TMS mini HTML engine, please refer to the paragraph especially devoted to this. To add items to the list, the Control.Items stringlist can be used. The HTMListbox descends from the VCL listbox and as such, all TListbox properties, methods and events can be used.

Additional properties of the THTMListbox / THTMLCheckList

AutoFocus: Boolean

When true, the listbox automatically gains focus when the mouse hovers the control.

AutoItemHeight: Boolean

When true, the height of the listbox item automatically adapts to the size of the text. This means that no horizontal scrollbar will be used for items that exceed the listbox width but instead the height of the item increases to fit the text. Note though that it is not supported to have items with a height equal to or higher than the listbox height itself. (This is actually a limitation of the underlying Windows listbox control)

Ellipsis: Boolean

When true, text that exceeds the width of the listbox will be drawn with an ellipsis (…) at the end. When ellipsis is true, text is always drawn on a single line.

EnableBlink: boolean

When the property EnableBlink is set to true, items that have blinking text set with the mini HTML <BLINK> tag will be drawn blinking.

HTMLHint: Boolean

When true in combination with ItemHint = true, the hint is automatically set to the HTML formatted text of the item. When HTMLHint is false, the hint text is set to the text of the listbox item with all HTML formatting tags removed. When HTMLHint is true, in order for the hint to be rendered with HTML formatting, it is required that the THTMLHint component is placed on the form. It is the THTMLHint component that will take care of overriding the default hint rendering and render with HTML formatting instead.

IncrLookup: Boolean

A regular VCL listbox supports keyboard lookup, this means that when a character is pressed when the listbox has focus, the listbox will select the first item that starts with the character. The THTMListbox also supports this lookup. Note that the lookup is performed based on the item text with all its tags removed. When IncrLookup is set to true, the THTMListbox has the additional feature that it performs an incremental lookup, ie. as characters are typed, the selected item is set to the item that starts with all characters pressed. When no more item is matching, the lookup is reset, ie. lookup is performed on new characters pressed.

LineSpacing: integer

This sets the spacing (in pixels) between two lines of HTML formatted text.

ItemHint: Boolean

When ItemHint is set to true, the hint of the listbox is automatically set to the text of the item over which the mouse hovers. This can be a convenient feature when the text of the item exceeds the width of the listbox and the hint can show the full text when the mouse hovers the item.

ScrollHorizontal: Boolean

When true, the listbox displays a horizontal scrollbar (when needed)

SelectionColors: TGradientStyle

This holds 4 colors and a border color to set the colors of the selected item(s). Note that the colors can be automatically set to predefined values by calling the method SetComponentStyle(). More information about styles can be found in the chapter specifically devoted to TAdvFormStyler, TAdvAppStyler.

ShadowOffset: integer / ShadowColor: TColor

These properties control the offset of the shadow and color of the shadow for text that has a shadow specified through the mini HTML tag <SHAD>

SortWithHTML: Boolean

When true, the HTML formatting tags are taken in account when performing a sort in the listbox. Otherwise, the sort is based on the text with all HTML formatting tags removed.

URLColor: TColor

This sets the color of hyperlinks in the HTML formatted text.

TextItems[index: integer]: string

This public property returns the text of an item with all its HTML formatting tags removed.

Additional events for the THTMListbox / THTMLCheckList

OnAnchorClick: This event is triggered when a hyperlink in the HTML formatted text is clicked. The event returns the index of the item that has the hyperlink and the HREF value of the hyperlink.

OnAnchorEnter: This event is triggered when the mouse enters the area of a hyperlink.

OnAnchorExit: This event is triggered when the mouse leaves the area of a hyperlink.
OnAnchorHint: This event is triggered when the mouse hovers a hyperlink and allows to dynamically set a hint specifically for a hyperlink. The event returns the index of the item in the listbox that has the hyperlink and the HREF value of the link. By default, this HREF value will be displayed as hint. By changing the variable parameter Anchor, the hint can be dynamically changed.

Example:

```
// This adds an item with hyperlink to the listbox
HTMListbox.Items.Add('Please click the link to visit <a
href="http://www.microsoft.com">Microsoft');
```

In order to not display the full URL in the hint when the mouse hovers the hyperlink, the OnAnchorHint event is used in following way:

```
procedure TForm1.HTMListBox1AnchorHint(Sender: TObject; Index: Integer;
  var Anchor: string);
begin
  if (Index = 0) then
    Anchor := 'Microsoft website';
end;
```

Additional methods for the THTMListbox / THTMLCheckList

HilightInList

The function HilightInList will show all parts of text matching a specific keyword as selected. The 2nd parameter DoCase specifies whether the keyword match is case sensitive or not.

HiLightInItem

The function HilightInItem is similar to HilightInList, except that it marks the text only in the item specified. The third parameter DoCase specifies whether the keyword match is case sensitive or not.

MarkInList

The function MarkInList will show all parts of text matching a specific keyword with a red curly underline. The 2nd parameter DoCase specifies whether the keyword match is case sensitive or not.

MarkInItem

The function MarkInItem is similar to MarkInList, except that it marks the text only in the item specified. The third parameter DoCase specifies whether the keyword match is case sensitive or not.

UnHilightInList

The function UnHilightInList will undo any previous call to HilightInList

UnHiLightInItem

The function UnHilightInItem will undo any previous call to HilightInItem

UnMarkInList

The function UnMarkInList will undo any previous call to MarkInList

UnMarkInItem

The function UnMarkInItem will undo any previous call to MarkInItem

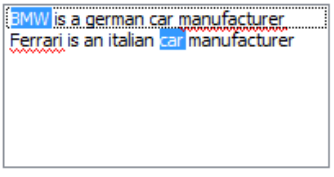
Example:

```
// This adds an item with hyperlink to the listbox
HTMListbox.Items.Add('Please click the link to visit <a
href="http://www.microsoft.com">Microsoft');
```

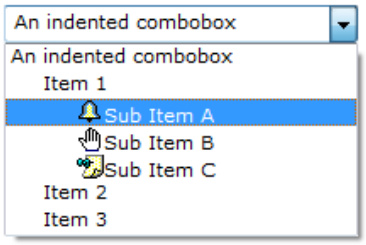
In order to not display the full URL in the hint when the mouse hovers the hyperlink, the OnAnchorHint event is used in following way:

```
procedure TForm1.HTMListBox1AnchorHint(Sender: TObject; Index: Integer;
  var Anchor: string);
begin
  if (Index = 0) then
    Anchor := 'Microsoft website';
end;
```

This results in:



THTMLComboBox



THTMLComboBox description

A combobox that can contain text with HTML formatting capabilities.

THTMLComboBox features

A combobox with HTML formatting capabilities. Supports a subset of the HTML formatting tags.

With HTML aware lookup and sorting. Combobox can be flat style. Supports the PictureContainer for adding GIF, JPEG, BMP, ICO or WMF files to the combobox.

THTMLComboBox use

The THTMLComboBox is a very similar extension for the regular VCL TComboBox to the THTMListBox. The THTMLComboBox is also a combobox where each item in the combobox can have HTML formatted text. Note that in order to have text display with HTML formatting in the edit control part of the combobox, it is required to set Style to csOwnerDrawFixed (this is the default setting). For full details about the support mini HTML tags, please refer to the paragraph specifically devoted to mini HTML formatting. The HTMLComboBox descends from the VCL combobox and as such, all TComboBox properties, methods and events can be used.

Additional properties of THTMLComboBox

AutoFocus:Boolean

When true, the combobox automatically gets focus when the mouse hovers the control

ButtonWidth: integer

Sets the width of the dropdown button when the combobox is set in flat mode (property Flat = true)

DropWidth:integer / DropHeight: integer

Sets the size of the dropdown part of the combobox

EditHeight: integer

Sets the height of the edit control part of the combobox

Ellipsis: Boolean

When true, text that exceeds the width of the listbox will be drawn with an ellipsis (…) at the end. When ellipsis is true, text is always drawn on a single line.

Flat: Boolean

When true, the combobox looks like a flat control, without a border and with a classic Windows dropdown button.

FocusBorder:Boolean

When true and when the property Flat is set to true, the combobox has a border when the control has focus.

HTMLHint: Boolean

When true in combination with ItemHint = true, the hint is automatically set to the HTML formatted text of the item. When HTMLHint is false, the hint text is set to the text of the combobox item with all HTML formatting tags removed. When HTMLHint is true, in order for the hint to be rendered with HTML formatting, it is required that the THTMLHint component is placed on the form. It is the THTMLHint component that will take care of overriding the default hint rendering and render with HTML formatting instead.

IncrLookup: Boolean

A regular VCL listbox supports keyboard lookup, this means that when a character is pressed when the listbox has focus, the listbox will select the first item that starts with the character. The THTMListbox also supports this lookup. Note that the lookup is performed based on the item text with all its tags removed. When IncrLookup is set to true, the THTMListbox has the additional feature that it performs an incremental lookup, ie. as characters are typed, the selected item is set to the item that starts with all characters pressed. When no more item is matching, the lookup is reset, ie. lookup is performed on new characters pressed.

SelectionColors: TGradientStyle

This holds 4 colors and a border color to set the colors of the selected item(s). Note that the colors can be automatically set to predefined values by calling the method SetComponentStyle(). More information about styles can be found in the chapter specifically devoted to TAdvFormStyler, TAdvAppStyler.

ShadowOffset: integer / ShadowColor: TColor

These properties control the offset of the shadow and color of the shadow for text that has a shadow specified through the mini HTML tag <SHAD>

URLColor: TColor

This sets the color of hyperlinks in the HTML formatted text.

Additional events of the THTMCombobox

OnAnchorClick

This event is triggered when a hyperlink in the HTML formatted text is clicked. The event returns the index of the item that has the hyperlink and the HREF value of the hyperlink.

OnAnchorEnter

This event is triggered when the mouse enters the area of a hyperlink.

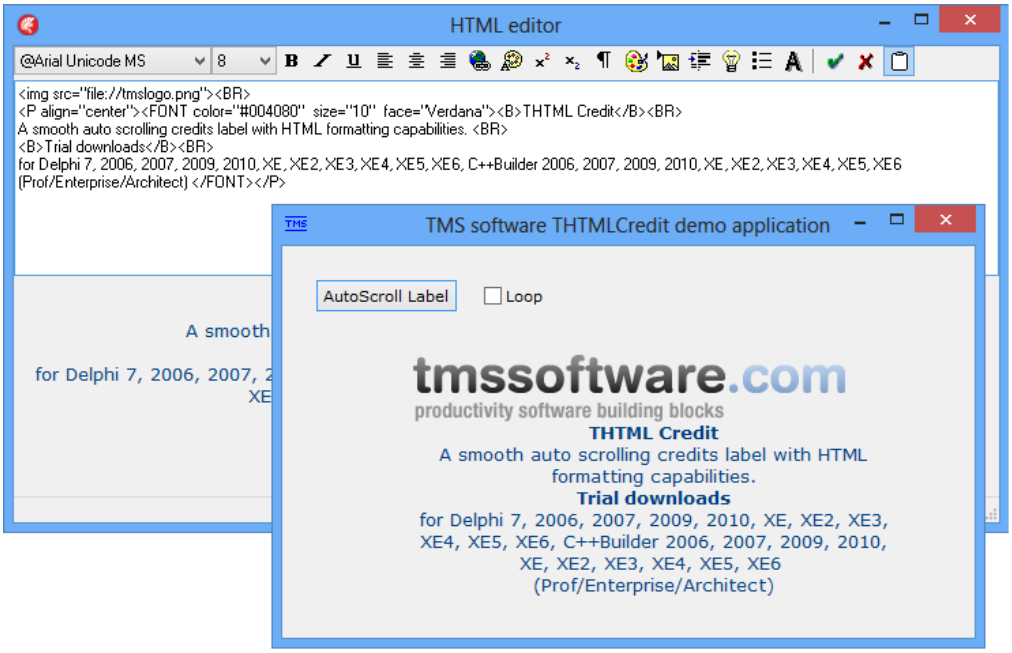
OnAnchorExit

This event is triggered when the mouse leaves the area of a hyperlink.

Additional methods of the THTMComboBox

The additional methods HilightInList, HilightInItem, MarkInList, MarkInItem, … are identical to these functions in the THTMListbox. Please refer to the THTMListbox additional methods description.

THTMLCredit



THTMLCredit description

Automatic scrolling credits label with HTML formatting capabilities.

THTMLCredit features

A smooth auto scrolling credits label with HTML formatting capabilities. Supports a subset of the HTML formatting.
Included support for bevel borders, events for anchor clicks, hovering, keyboard interface for anchors, blinking text, GIF, JPEG, ICO, BMP, WMF, imagelist images, mini scrollbar, various autosizing styles and a HTML property editor ...

THTMLCredit use

Setting HTMLCredit text

Set the text to show in the THTMLCredit component via the HTMLText property. This is a TStrings property via which you can add text line by line using : HTMLCredit.HTMLText.Add('another line') or you can set the text directly with HTMLCredit.HTMLText.Text := 'This is all the text for HTMLCredit'. The text for the HTMLCredit component can use HTML formatting. Refer to the chapter devoted to the TMS mini HTML engine for all supported tags.

Auto sizing & scrolling

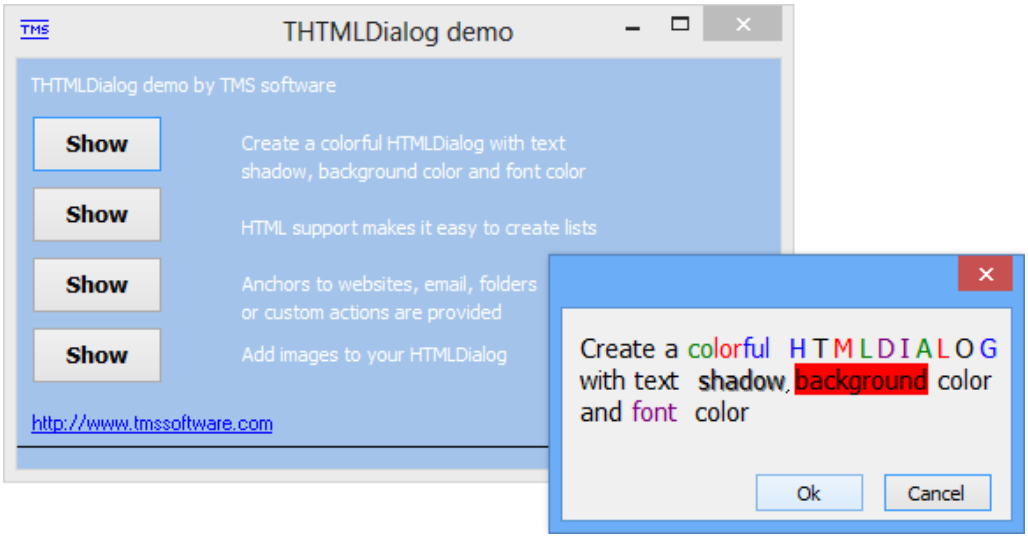
The size of the HTMLCredit component can automatically adapt to the text set in HTMLText. To enable this, set HTMLCredit.AutoSize = true. With the property AutoSizeType, it can be controlled if the auto sizing is applied to the width of the component, the height or both width and height.
When the property HTMLCredit.AutoScroll is set to true, the component performs automatic upwards scrolling of the HTML formatting text (like typical scrolling credits). The scrolling can further be finetuned with properties:

ScrollSpeed: sets the time in milliseconds between scroll position updates
ScrollSteps: sets the number of pixels to scroll up for each scroll position update
The scroll postion can also be programmatically set (instead of letting the component perform auto scrolling) with the property HTMLCredit.ScrollPosition.

Anchors

For anchors (hyperlinks) in the HTMLCredit component, the events OnAnchorClick, OnAnchorEnter and OnAnchorExit are available. The hyperlinks are displayed by default in the color URLColor. An additional feature is available to let the hyperlinks show in a different background and text color when the mouse hovers the hyperlink. To enable this, set HTMLCredit.Hover = true and define the color to use for the hover hyperlink background and text color with HTMLCredit.HoverColor and HTMLCredit.HoverFontColor.

THTMLDialog



THTMLDialog description

Message Dialog with HTML formatting capabilities.

THTMLDialog features

- Message dialog with HTML formatting capabilities
- All HTML formatting can be set at design time with HTML preview property editor
- Requires THTMLabel to be installed

THTMLDialog use

The THTMLDialog offers an alternative to standard dialogs allow to display much richer HTML formatted text that can include images, hyperlinks, …

Setting THTMLDialog text

Set the text to show in the THTMLDialog component via the HTMLText property. This is a TStrings property via which you can add text line by line using : HTMLDialog.HTMLText.Add('another line') or you can set the text directly with HTMLDialog.HTMLText.Text := 'This is all the text for HTMLDialog'. The text for the HTMLDialog component can use HTML formatting. Refer to the chapter devoted to the TMS mini HTML engine for all supported tags.

Dialog buttons

The HTMLDialog can have up to 3 buttons: a left button, center button and right button. Properties for these buttons are available via HTMLDialog.LeftButton, HTMLDialog.CenterButton, HTMLDialog.RightButton. For each button following properties can be set:

Cancel: boolean: When true the button represents the cancel action (ESC)
Caption:string: caption text of the button
Default: Boolean: When true, the button represents the default action (Return)
ModalResult: TModalResult: sets the modal result when the button is clicked
Visible: Boolean: Sets the button visibility.

Anchors

For anchors (hyperlinks) in the HTMLDialog component, the events OnAnchorClick, OnAnchorEnter and OnAnchorExit are available. The hyperlinks are displayed by default in the color URLColor. An additional feature is available to let the hyperlinks show in a different background and text color when the mouse hovers the hyperlink. To enable this, set HTMLDialog.Hover = true and define the color to use for the hover hyperlink background and text color with HTMLDialog.HoverColor and HTMLDialog.HoverFontColor.

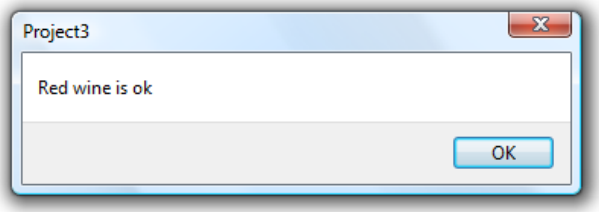
Showing the dialog

To show the dialog, set its properties and call HTMLDialog.ShowModal.

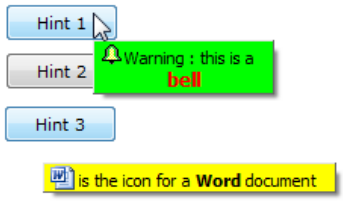
This code snippet shows the dialog and handles the result:

```
begin
  HTMLDialog1.HTMLText.Text := 'Do you prefer <b><font
color="clred">red</font></b> wine with your dinner?';
  HTMLDialog1.CenterButton.Visible := true;
  HTMLDialog1.CenterButton.Caption := 'Yes';
  HTMLDialog1.CenterButton.ModalResult := mrYes;
  HTMLDialog1.CenterButton.Default := true;
  HTMLDialog1.RightButton.Visible := true;
  HTMLDialog1.RightButton.Caption := 'No';
  HTMLDialog1.RightButton.ModalResult := mrNo;
  HTMLDialog1.RightButton.Cancel := true;
  if HTMLDialog1.ShowModal = mrYes then
    ShowMessage('Red wine is ok');
end;
```

Result:



THTMLHint



THTMLHint description

Gives your applications HTML formatted hints.

THTMLHint features

Drop a HTMLHint component on the main form of your application and you can start using HTML formatted hints everywhere in your application. THTMLHint supports a subset of the HTML formatting.

THTMLHint use

THTMLHint window hooks into the Delphi application hint system. It effectively replaces the standard Delphi hints and performs its own rendering. This means that for any component's Hint property, HTML formatting tags can be used. The HTML formatted hint of a component can contain references to images (<IMG> tag) and the THTMLHint can render these via its attached ImageList or PictureContainer.

Example:

```
Button.Hint := '<IMG src="idx:1">This is a hint with <b>HTML</b>
formatting';
Button.ShowHint : =true;
```

With this hint set for the button, when THTMLHint is used in the application, the hint will show with HTML formatting rendered and the image with index 1 of the ImageList attached to THTMLHint.Images will be shown.

Size of the hint

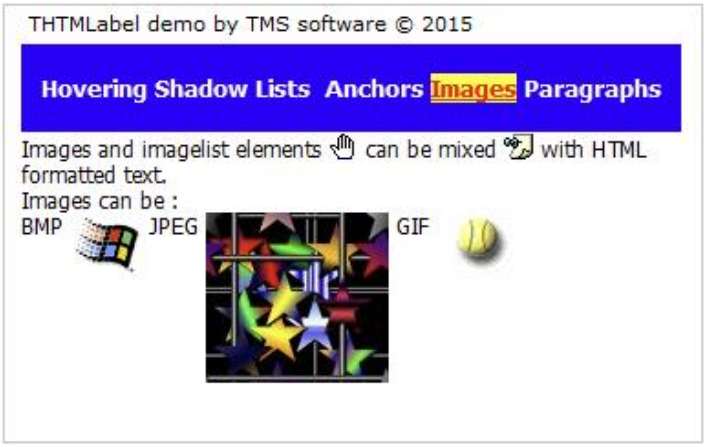
Normally, the size of the hint automatically adapts to the text defined for the hint. The sizing of the hint can be further controlled with the property HTMLHint.MaxWidth. When this is set to zero (default value), the hint width will be set as wide as necessary to fit the longest line of text for the hint. When MaxWidth is set to a value different from zero, the maximum width of the hint will be MaxWidth and instead the height of the hint will increase to fit the wordwrapped HTML formatted text. Alternatively, when HTMLHint.Ellipsis is set to true, the hint will always display as a single line hint and will display ellipsis (…) when the hint text exceeds the available width.

Additional visual control

HintColor: sets the gradient start color of the hint background
HintColorTo: sets the gradient end color of the hint background. When HintColorTo is set to clNone, the hint background is displayed with a solid background color HintColor

HintFont: sets the font for the hint
HintStyle: selects between a rectangle or a rounded rectangle
ShadowOffset: sets the offset of HTML text with a shadow (<SHAD> tag)
ShadowColor: sets the color of the HTML text with a shadow
URLColor: sets the color of hyperlinks in the hint. Note that hyperlinks are by design not clickable on the hint as the hint window is never an active window and can as such not respond to mouse clicks.
XMargin, YMargin: sets the horizontal & vertical margin between hint border and hint text.

THTMLabel



THTMLabel description

A label with HTML formatting capabilities

THTMLabel features

A label with extensive HTML formatting capabilities. Supports a subset of the HTML formatting tags. Included support for bevel borders + events for anchor clicks. Function to hilight and mark text.

THTMLabel use

THTMLabel is designed to be feature compatible with a standard TLabel but providing many extra capabilities to display HTML formatted text, gradient background, auto sizing and much more.

Setting THTMLabel text

Set the text to show in the THTMLabel component via the HTMLText property. This is a TStringList property via which you can add text line by line using : HTMLabel.HTMLText.Add('another line') or you can set the text directly with HTMLabel.HTMLText.Text := 'This is all the text for HTMLabel'. The text for the HTMLabel component can use HTML formatting. Refer to the chapter devoted to the TMS mini HTML engine for all supported tags.

Auto sizing

The size of the HTMLabel component can automatically adapt to the text set in HTMLText. To enable this, set HTMLabel.AutoSize = true. With the property AutoSizeType, it can be controlled if the auto sizing is applied to the width of the component, the height or both width and height.

Anchors

For anchors (hyperlinks) in the HTMLabel component, the events OnAnchorClick, OnAnchorEnter and OnAnchorExit are available. The hyperlinks are displayed by default in the color URLColor. An additional feature is available to let the hyperlinks show in a different background and text color when the mouse hovers the hyperlink. To enable this, set HTMLabel.Hover = true and define the color to use for the hover hyperlink background and text color with HTMLabel.HoverColor and HTMLabel.HoverFontColor.

When THTMLabel.ShowHint is set to true, a hint is also displayed for anchors. By default, the value for a hint of an anchor is set to the HREF attribute. To override this, the OnAnchorHint event can be used. This returns the HREF value as a parameter but via this event, the hint text for a hyperlink can be dynamically changed.

HTMLabel background

By default, the THTMLabel background is transparent. The THTMLabel.Transparent: Boolean property sets this. When Transparent is set to false, the background can be a solid color or a gradient. The gradient start and end color is set with HTMLabel.Color and HTMLabel.ColorTo. Further capabilities of the gradient can be selected with HTMLabel.GradientType:

- gtFullHorizontal: a horizontal gradient fills the entire background
- gtFullVertical: a vertical gradient fills the entire background
- gtCenterLine: a gradient line is drawn vertically centered in the label
- gtB ottomLine: a gradient line is drawn top aligned in the label
- gtTopLine: a gradient line is drawn bottom aligned in the label

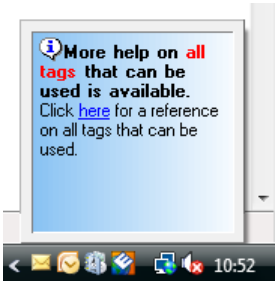
THTMLabel tips and FAQ

How to use the design-time mini HTML editor at run-time

You can easily use the mini HTML design-time editor at run-time to allow users to edit the mini-HTML controls text at run-time. To allow this, use following steps : add the HTMLPROP unit to your project and add HTMLPROP to the uses clause in the unit from where you want to invoke the mini HTML editor. Add the following code to allow for example run-time editing of the content of a HTMLabel component:

```
var
  htmledit: THTMLEditor;
begin
  htmledit := THTMLEditor.Create(Self);
  try
    htmledit.memo1.lines.Text := htmlabel1.htmltext.Text;
    htmledit.ShowModal;
    htmlabel1.htmltext.Text := htmledit.memo1.lines.Text;
  finally
    htmledit.Free;
  end;
end;
```

THTMLPopup



THTMLPopup description

An MS Messenger like popup box with HTML formatting capabilities.

THTMLPopup features

A MS Messenger like popup box that can roll up & down from the bottom of the screen with HTML formatting capabilities. Supports a subset of the HTML formatting tags.

THTMLPopup use

Using THTMLPopup is straightforward. The HTML formatted text to display in the popup window can be set with HTMLPopup.Text stringlist property. The size of the popup window is set with HTMLPopup.PopupWidth, HTMLPopup.PopupHeight.

Showing the popup window

The popup window can be displayed at any arbitrary position on the screen or can be animated with a roll up effect from the bottom right of the screen (near the taskbar icons). To show the popup window at a specific location, use the PopupLeft, PopupRight properties to set in screen coordinates the position of the popup window and call the method Show.

Example:

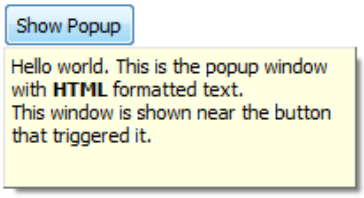
This code displays a HTMLPopup window at the bottom of a button on the form:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  pt: TPoint;
begin
  HTMLPopup1.Text.Clear;
  HTMLPopup1.Text.Add('Hello world. This is the popup window with
<b>HTML</b> formatted text.<br>');
  HTMLPopup1.Text.Add('This window is shown near the button that triggered
it.');
  HTMLPopup1.PopupHeight := 80;
  pt.X := Button1.Left;
  pt.Y := Button1.Top + Button1.Height;

  pt := ClientToScreen(pt);
```

```
  HTMLPopup1.PopupLeft := pt.X;
  HTMLPopup1.PopupTop := pt.Y;
  HTMLPopup1.show;
end;
```

This results in:



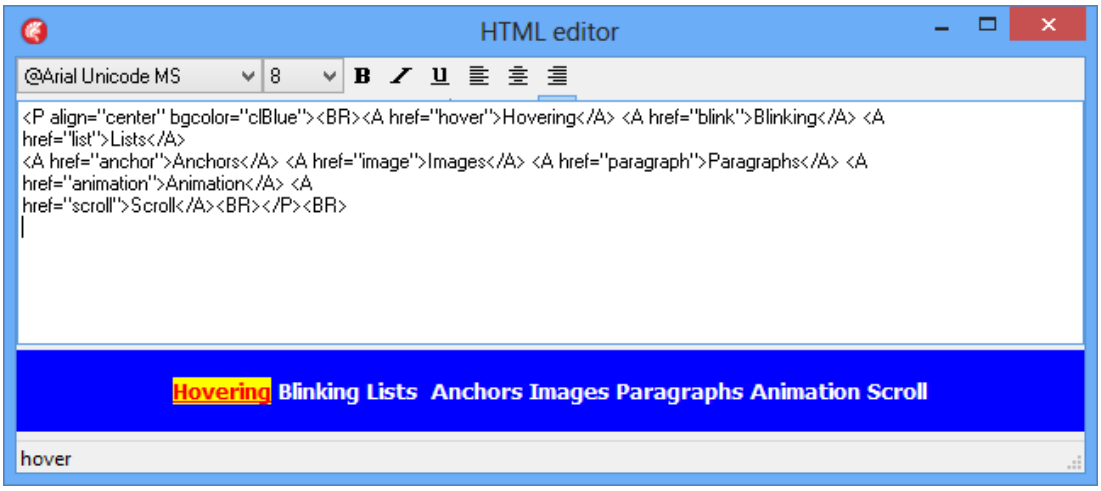
Alternatively, the popup window can be animated with a roll up effect from the bottom right of the screen by calling HTMLPopup.RollUp; One additional property RollUpSpace controls the roll up effect. This sets the space between the taskbar and the bottom of the HTMLPopup window (this is default zero)

If you want that the HTMLPopup window shows on top of all other windows, irrespective of the application being active or not, set HTMLPopup.AlwaysOnTop to true. Normally, the HTMLPopup window is visible till HTMLPopup.Hide is called. When HTMLPopup.AutoHide is true, the HTMLPopup window hides as soon as the mouse leaves the window.

Auto sizing

The size of the HTMPopup window can automatically adapt to the text set in HTMLText. To enable this, set HTMLabel.AutoSize = true. With the property AutoSizeType, it can be controlled if the auto sizing is applied to the width of the component, the height or both width and height.

THTMLStaticText



THTMLStaticText description

A label with HTML formatting capabilities.

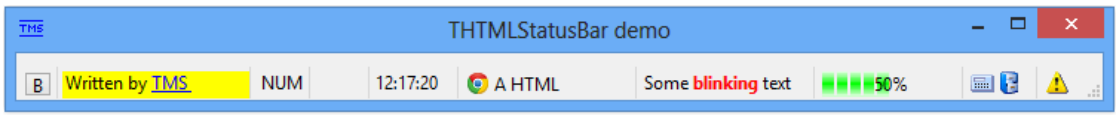
THTMLStaticText features

A label with HTML formatting capabilities. Supports a subset of the HTML formatting. Included support for bevel borders, events for anchor clicks, hovering, keyboard interface for anchors, blinking text, GIF, JPEG, ICO, BMP, WMF, imagelist images, mini scrollbar, various autosizing styles and a HTML property editor ...

THTMLStaticText use

The THTMLStaticText is functionally very similar to the THTMLabel component. Main difference is that the THTMLabel descends from a TGraphicControl and THTMLStaticText descends from a TCustomControl. This means that THTMLabel cannot have focus. THTMLStaticText can have keyboard focus and as such has a TabStop / TabOrder property.
In addition to the THTMLabel, the THTMLStaticText component has the extra property HTMLStaticText.EnableBlink. When this property is set to true, text in the label between the tags <BLINK> </BLINK> will be rendered as blinking text.

THTMLStatusBar



THTMLStatusBar description

A statusbar with panels that can contain text with HTML formatting & more.

THTMLStatusBar features

- support for mini HTML formatted text in panels, including capability to add hyperlinks, images and blinking text in the statusbar panel
- panel with time, date, NUMLock, CAPSLock, SCROLLLock, image, imagelist, animated image, ellips text, file ellips text
- panel with progress bar with 4 level stacked, smooth, XP style look
- accepts other controls on the statusbar

THTMLStatusBar use

THTMLStatusBar is a statusbar that supports displaying HTML formatting text in a statuspanel but offers at the same time a lot more features.

Panels collection and properties

Just like a regular statusbar, the panels within a statusbar are setup via the collection: Panels. This collection contains instances of the class THTMLStatusPanel that has following properties:

Alignment: Boolean : sets the alignment of the text within the statusbar panel
Animated: Boolean : when true, the panel shows an animation of imagelist images.
AnimationDelay: integer : this sets the delay in milliseconds between two images of the AnimationImages ImageList for panels configured as animation of images.
Bevel: THTMLStatusPanelBevel : this sets the bevel of the panel
BidiMode : TBiDiMode : this sets the Bidi mode of the panel
DateFormat: string : sets the format for the date displayed in the panel when the panel style is psDate.
Enabled: Boolean : sets the panel as enabled when true
Hint :string : sets the hint for the panel
HTMLOffsetX: integer : sets the offset from left side of the panel where HTML formatted text starts.
HTMLOffsetY: integer : sets the offset from the top of the panel where the HTML formatted text starts.
ImageIndex: integer : sets the index of the image from the imagelist for a panel with style psImage.
ParentBidiMode: Boolean : when true, the BidiMode is set to the BidiMode of the parent control.
Progress: TProgressStyle : holds the various color & appearance settings for a progress bar used in the panel.
Style: THTMLStatusPanelStyle : sets the style of the panel.
Text: string : sets the text of the statusbar panel.
TimeFormat: string : sets the format for the time displayed in the panel when the panel style is psTime.

Width: integer : sets the width of the panel.

Panel styles

Following panel styles are built-in:

psHTML: panel displays HTML formatted text
psText: panel displays text
psOwnerDraw: panel is owner drawn via the OnDrawPanel event
psTime: panel displays the current system time formatted with the TimeFormat format
psDate: panel displays the current system date formatted with the DateFormat format
psNumLock: panel displays the state of the NumLock key
psCapsLock: panel displays the state of the CapsLock key
psScrollLock: panel displays the state of the ScrollLock key
psProgress: panel displays a progress bar
psImage: panel displays an image from the imagelist
psImageList: panel displays multiple images from an imagelist
psAnimation: panel displays an animated sequence of imahes
psEllipsText: panel displays text with end ellipsis
psFileEllipsText: panel displays text with file ellipsis

Progressbar styles

Sophisticated progressbar types are supported in the HTMLStatusBar panels. The progressbars support multiple colors depending on the level of the progressbar. The TProgressStyle persistent class controls how the progressbar is displayed:

Background: TColor: sets the background color of the progressbar

BorderColor: TColor: sets the border color of the progressbar

CompletionSmooth: Boolean: when false, the progressbar consists of blocks, when true it is one continuous line

Indication: TProgressIndication: defines whether the indication of the position is as percentage (piPercentage) or an absolute number (piAbsolute)

Level0Color: TColor: Sets the gradient start color for progressbar positions below value Level0Perc

Level0ColorTo: TColor: Sets the gradient end color for progressbar positions below value Level0Perc. When Level0ColorTo is clNone, a solid color is used.

Level1Color: TColor: Sets the gradient start color for progressbar positions below value Level1Perc and above Level0Perc

Level1ColorTo: TColor: Sets the gradient end color for progressbar positions below value Level1Perc and above Level0Perc. When Level1ColorTo is clNone, a solid color is used.

Level2Color: TColor: Sets the gradient start color for progressbar positions below value Level2Perc and above Level1Perc.

Level2ColorTo: TColor: Sets the gradient end color for progressbar positions below value Level2Perc and above Level1Perc. When Level3ColorTo is clNone, a solid color is used.

Level3Color: TColor: Sets the gradient start color for progressbar positions at 100%.

Level3ColorTo: TColor: Sets the gradient end color for progressbar positions at 100%. When Level3ColorTo is clNone, a solid color is used.

Max: integer: Maximum value of the progress position

Min: integer: Minimum value of the progress position

Position: integer: Current position of the progressbar

Prefix: string: prefix to use for the progress position value text displayed

ShowBorder: Boolean: Shows a border around the progressbar when true

ShowGradient: Boolean: Uses a gradient for progressbar colors when true

Stacked: Boolean: When true, the different level colors are shown stacked

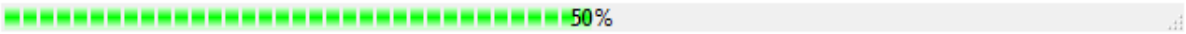
Suffix: string: suffix to use for the progress position value text displayed

TextColor: TColor: color of the text on the progressbar

Examples:

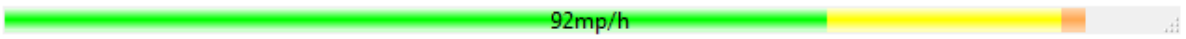
This code snippet adds a progress style panel to the status bar in default style:

```
begin
  HTMLStatusBar1.Panels.Clear;
  HTMLStatusBar1.Panels.Add;
  HTMLStatusBar1.Panels[0].Style := psProgress;
  HTMLStatusBar1.Panels[0].Progress.Position := 50;
end;
```

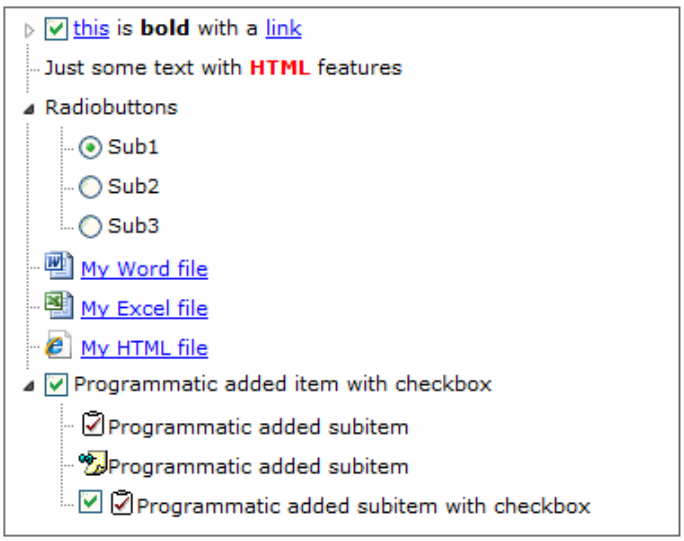


This code snippet adds a progress style panel to the status bar with a stacked progressbar with absolute value indication and suffix set to ‘mp/h’:

```
begin
  HTMLStatusBar1.Panels.Clear;
  HTMLStatusBar1.Panels.Add;
  HTMLStatusBar1.Panels[0].Style := psProgress;
  HTMLStatusBar1.Panels[0].Progress.Position := 92;
  HTMLStatusBar1.Panels[0].Progress.Stacked := true;
  HTMLStatusBar1.Panels[0].Progress.Indication := piAbsolute;
  HTMLStatusBar1.Panels[0].Progress.Suffix := 'mp/h';
  HTMLStatusBar1.Panels[0].Progress.CompletionSmooth := true;
end;
```



THTMLTreeView



THTMLTreeView description

A treeview and listview combined with columns with different formatting capabilities.

THTMLTreeView features

A treeview with HTML formatting capabilities, radiobutton and checkbox capability. Checkboxes and radiobuttons support various looks, from standard, flat, Borland style to Windows XP visual styles. Supports a subset of the HTML formatting tags (see the mini HTML reference).

THTMLTreeView use

The THTMLTreeview is very similar in use to the standard VCL TTreeview, except that the text for every node can contain HTML formatting. As such, the THTMLTreeview can be used in the same way as a TTreeview but the extra capability to add HTML tags to the text for a TTreeNode is offered. Refer to the chapter devoted to the mini HTML engine for details about all HTML tags that are supported.

Anchors

For anchors (hyperlinks) in the THTMLTreeview component, the events OnAnchorClick, OnAnchorEnter and OnAnchorExit are available. The hyperlinks are displayed by default in the color URLColor.

Checkboxes & radiobuttons

Every node in the treeview can have a checkbox or radiobutton. The functions in THTMLTreeview to work with checkboxes & radiobuttons in the nodes are:

```
SetNodeCheck(Node: TTreeNode; Checked: Boolean);
```

This method adds a checkbox initialized with its state to Checked to the node Node.

```
GetNodeCheck(Node: TTreeNode; var Checked: Boolean);
```

This function retrieves the checkbox state of the checkbox in node Node.

```
RemoveNodeCheck(Node: TTreeNode);
```

This method removes the checkbox from node Node.

```
SetRadioButton(Node: TTreeNode; Checked: Boolean);
```

This method adds a radiobutton initialized with its state to Checked to the node Node.

```
GetRadioButton(Node: TTreeNode; var Checked: Boolean);
```

This function retrieves the radiobutton state of the radiobutton in node Node.

```
RemoveRadioButton(Node: TTreeNode);
```

This method removes the radiobutton from node Node.

Note that for radiobuttons, all radiobuttons added on the same node level form a radiogroup, i.e. clicking one node will automatically uncheck the radiobutton of nodes at the same level.

The checkbox and radiobutton trigger an event when clicked. For a click on the checkbox, the OnCheckBoxClick event is triggered, returning the Node clicked and the checkbox state. For the radiobutton, the OnRadioClick event is triggered returning the Node clicked.

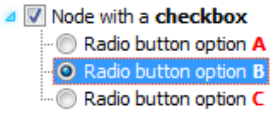
Example:

This code adds a checkbox and three radiobuttons to the HTMLTreeview and handles the click:

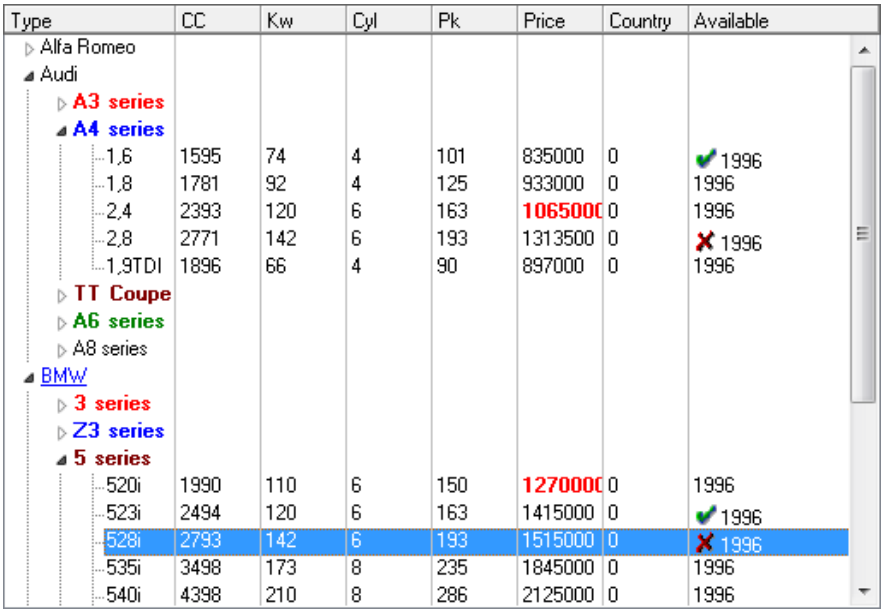
```
procedure TForm1.InitTreeview;
var
  tn,cn: TTreeNode;
begin
  tn := HTMLTreeView1.Items.Add(nil,'Node with a <b>checkbox</b>');
  HTMLTreeview1.SetNodeCheck(tn,false);
  cn := HTMLTreeview1.Items.AddChild(tn,'Radio button option <b><font
color="clred">A</b>');
  HTMLTreeview1.SetRadioButton(cn,false);
  cn := HTMLTreeview1.Items.AddChild(tn,'Radio button option <b><font
color="clred">B</b>');
  HTMLTreeview1.SetRadioButton(cn,false);
  cn := HTMLTreeview1.Items.AddChild(tn,'Radio button option <b><font
color="clred">C</b>');
  HTMLTreeview1.SetRadioButton(cn,false);
end;
procedure TForm1.HTMLTreeview1CheckBoxClick(Sender: TObject; Node:
TTreeNode;
  Check: Boolean);
begin
  if check then
    ShowMessage('click checkbox: checked')
  else
    ShowMessage('click checkbox: not checked');
end;
```

```
procedure TForm1.HTMLTreeview1RadioClick(Sender: TObject; Node: TTreeNode);
begin
  ShowMessage('click radiobutton: checked:'+ Node.Text)
end;
```

This results in:



THTMLTreeList



THTMLTreeList description

A treeview with HTML formatting capabilities.

THTMLTreeList features

A treeview with columns that can have width, alignement and column header specified. This is possible through a new introduced Columns property.
The item height can be specified through a new property ItemHeight. Each node can have HTML formatted text in each column.

THTMLTreeList use

THTMLTreeList adds a multicolumn structure to the capabilities of the THTMLTreeview. Via the property Columns, the columns for the treelist can be defined. For each column, the font, header text and width can be set.

The text for the different columns for each node is set via TTreeNode.Text. Text for separate columns is separated by a separator character. By default, the separator character is ';'. To override this, the public property THTMLTreeList.Separator can be used.

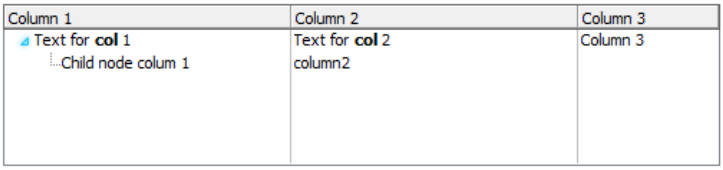
Example:

This code snippet shows how the add three columns to the THTMLTreeList and add nodes with text for the three columns. The column separator is forced to the character '#' here:

```
var
  tn: TTreeNode;
begin
  HTMLTreeList1.Width := 500;
```

```
  HTMLTreeList1.Columns.Clear;
  HTMLTreeList1.Columns.Add.Width := 200;
  HTMLTreeList1.Columns.Add.Width := 200;
  HTMLTreeList1.Columns.Add.Width := 100;
  HTMLTreeList1.Columns[0].Header := 'Column 1';
  HTMLTreeList1.Columns[1].Header := 'Column 2';
  HTMLTreeList1.Columns[2].Header := 'Column 3';
  HTMLTreeList1.Separator := '#';
  tn := HTMLTreeList1.Items.Add(nil,'Text for <b>col</b> 1#Text for
<b>col</b> 2#Column 3');
  HTMLTreeList1.Items.AddChild(tn,'Child node colum 1#column2');
end;
```

This results in:



TDBHTMLabel

TDBHTMLabel description

A data-aware label with HTML formatting capabilities.

TDBHTMLabel features

A DB-aware label with full mini-HTML formatting capabilities that can be linked to multiple DB fields with DB-aware design time HTML editor.
A label with HTML formatting capabilities. Supports a subset of the HTML formatting.
Included support for bevel borders + events for anchor clicks.

TDBHTMLabel use

The TDBHTMLabel descends from the THTMLabel and has the extra feature to support databinding to one or more dataset fields via HTML tags. As such, it is possible to define where to display dataset field data in the HTML formatted text. This is achieved by specifying the dataset field in the HTML with a tag: <#FIELDNAME>. All occurrences of tags <#FIELDNAME> are on-the-fly replaced by their dateset field value. To do this, the TDBHTMLabel will internally use dataset.FieldByName(FieldName).DisplayText to retrieve the value to display.
The dataset field can not only contain text but it is equally possible to show dataset blob field images in the TDBHTMLabel. To do this, insert the tag in the IMG source attribute, ie: <IMG src="<#FIELDNAME>">.
In case a dataset field value cannot be obtained using Field.DisplayText, the event TDBHTMLabel.OnGetData can be used to add custom code to retrieve a text representation for a dataset field value.

Normally, the dataset field value is displayed with the formatting as defined in the dataset field itself. Further dynamic formatting or modifying of the dataset field value before displaying is also supported via the event OnTransformData. In this sample code snippet, a city code is transformed to a full city name for display in the TDBHTMLabel. Assuming the TDBHTMLabel text is set with:

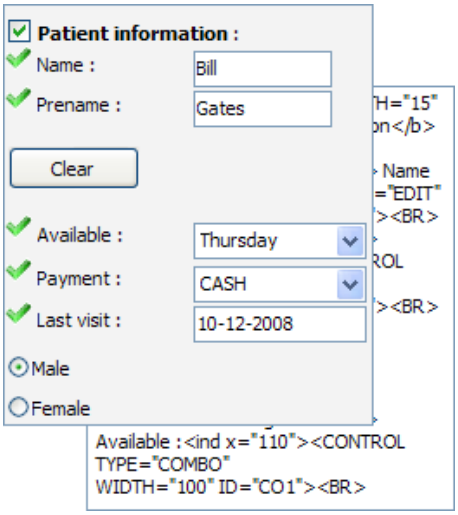
```
begin
  DBHTMLabel1.HTMLText.Text := 'Your flight to <B><#CITY></B>';
end;
procedure TForm1.DBHTMLabel1TransformData(Sender: TObject; Tag: string;
  var Data: string);
begin
   if Tag = 'CITY' then
   begin
     if Data = 'PAR' then
       Data := 'Paris'
```

```
      else
      if Data = 'LON' then
        Data := 'London'
      else
      if Data = 'BER' then
        Data := 'Berlin';
   end;
end;
```

The label will display

Your flight is to Paris
if the dataset field value was ‘PAR’.

THTMLForm



THTMLForm description

Create dynamic runtime HTML forms with controls

THTMLForm features

- Create forms with HTML
- Forms can use edit, checkbox, radiobutton, combobox, datetimepicker input controls
- Supports a subset of the HTML formatting tags (see the Mini HTML reference)
- Easy methods GetValue/SetValue to programmatically get and set form control values

THTMForm use

The THTMForm can be considered as THTMLabel that is able to display HTML formatted text in a rectangular area. But in addition to a THTMLabel, it can display & handle HTML specified controls such as edit control, combobox control, button, radiobutton and checkbox. This means that in addition to the regular mini HTML tags supported, the THTMLForm control supports the CONTROL tag with several options.

<CONTROL TYPE=”controltype” WIDTH=”controlwidth” ID=”controlid” [VALUE=”controlvalue”] [PROP=”controlprop”] [MAXLEN=”controlmaxlen”]>

The controltype can be:

“CHECK”: checkbox
“RADIO”: radiobutton
“EDIT”: edit control
“PASSWORD”: edit control in password style
“MASK”: edit control with input mask
“COMBO”: combobox
“BUTTON”: button

The controlwidth is the desired width of the control in pixels.

The controlid is a unique identifier of the control in the scope of the HTML of the THTMLForm component.
The controlvalue is the value displayed in the control.

The controlprop is an optional attribute that can set extra properties for the input control. It is not used with every input control. It is used for the masked input edit for example and contains the mask to be used. Note that the mask specification is the same as for a regular VCL TMaskEdit.

The controlmaxlen is an optional attribute that can set the max. accepted input length for an EDIT or PASSWORD input control.

Examples:

A 15pixel wide checkbox with ID “CK1”:

```
<CONTROL TYPE="CHECK" WIDTH="15" ID="CK1">
```

A radiogroup of 2 radiobutton items:

```
<CONTROL TYPE="RADIO" ID="RD1" Width="20">Male<br>
<CONTROL TYPE="RADIO" ID="RD2" Width="20">Female<br>
```

An 100 pixel wide edit control with preset value of “100”:

```
<CONTROL TYPE="EDIT" WIDTH="100" VALUE="" ID="ED1">
```

A 150 pixel wide combobox with ID “CO1”

```
<CONTROL TYPE="COMBO"  WIDTH="150" ID="CO1">
```

Getting and setting form control values

The THTMLForm component has following methods:

procedure HTMLForm.SetValue(ID, AValue: string);

This sets the value “AValue” for the control with identifier ID.

Example:

```
// sets a checkbox to true
  HTMLForm1.SetValue('CK1', 'true');
// sets an input edit value
  HTMLForm1.SetValue('ED1', 'Bill');
// sets a combobox value
  HTMLForm1.SetValue('CO1', 'Thursday');
// sets a radiobutton checked
  HTMLForm1.SetValue('RD1', 'true');
```

function GetValue(ID: srting; var AValue: string):boolean;

The function GetValue() returns true when the HTML input control with unique ID “ID” was found and returns the value of the control via the AValue parameter.

Example:

```
var
   s: string;
// gets a checkbox to true
  HTMLForm1.GetValue('CK1', s);
// gets an input edit value
  HTMLForm1.GetValue('ED1', s);
// gets a combobox value
  HTMLForm1.GetValue('CO1', s);
// gets a radiobutton checked
  HTMLForm1.GetValue('RD1', s);
```

THTMLForm events

The THTMLForm control provides several control related events that enable to programmatically interact with the controls.

OnControlClick(Sender: TObject; CtrlID, CtrlType, CtrlVal: string);

Event triggered when a control is clicked. It returns the unique ID of the control, its type and the vakue.

OnControlComboList(Sender: TObject; CtrlID, CtrlType, CtrlVal: string; Values: TStringList; var Edit: Boolean; var DropCount: Integer);

Event triggered when a combobox control in the THTMLForm is activated. Via this event, the values of the combobox are set and it is determined whether it is an editable or not editable combobox, i.e. combobox with style csDropDown (Edit= true) or csDropDownList (Edit = false). Via the Values stringlist parameter, the combobox values can be set.

Example:

This event handler sets values for two comboboxes in the THTMLForm and sets one combobox as style csDropDown and the other with style csDropDownList:

```
procedure TForm1.HTMLForm1ControlComboList(Sender: TObject; CtrlID, CtrlType,
  CtrlVal: string; Values: TStringList; var Edit: Boolean;
  var DropCount: Integer);
begin
  Values.Clear;
  if CtrlID = 'CO1' then
  begin
   Values.Add('Monday');
   Values.Add('Tuesday');
   Values.Add('Wednesday');
   Values.Add('Thursday');
   Values.Add('Friday');
   Values.Add('Saturday');
   Values.Add('Sunday');
   Edit := False; // combo dropdownlist
  end;
```

```
  if CtrlID = 'CO2' then
  begin
   Values.Add('VISA');
   Values.Add('AMEX');
   Values.Add('MASTERCARD');
   Values.Add('CASH');
   Values.Add('N/A');
   Edit := True; // combo dropdown edit
  end;
end;
```

OnControlComboSelect(Sender: TObject; ItemIndex: Integer; CtrlID, CtrlValue: string);

Event triggered when a value in a THTMLForm combobox control is selected. It returns the ID of the combobox control and the index + value of the item selected.

OnControlEditDone(Sender: TObject; CtrlID, CtrlType, CtrlVal: string);

Event triggered when the focus leaves a THTMLForm input control. It returns the unique ID of the control that loses focus, its type and value.

OnControlEditMaskError(Sender: TObject; CtrlID, CtrlType, CtrlVal: string);

Event triggered when the focus leaves a THTMLForm edit input control that has an input mask and the entry did not match the input mask.