HTML Library

Table of Contents

1 Units

Core units

htmlpars - parsing and base node classes - THtNode, THtmlNode
htmlcss - CSS styles, TStyledHTMLNode, TCSSStyleSheet, etc.
htcanvas - base canvas class.
htdictionary - dictionaries
hthints - HTML hints
htinet - HTTP client for loading images and styles via HTTP
htmlani - CSS animations
htpdf - direct PDF export (requires Office library).
htplatform - platform depending classes.
htresource - localization.
htscriptIndy - Indy classes registration for Scripter
htscriptparse - Scripter
httemplates - template engine
htTextRenderer - classes for plain text canvas
htutils - common functions
htxml - XML and JSON parsing.

Following units has two versions depending on framework - VCL or FMX:

VCL:

htmldraw - TElement and THtDocument classes.
httables - HTML tables support - TTableElement, TTableCell, TTableRow, etc.
htsvg - SVG support - TSVGElement, TSVGPathElement, etc.
htclasses - additional element classes - TFrameElement, TInputElement, TSelectElement, etc.
htdefscriptadapter - script adapter class
htfontawesome - FontAwesome registration for VCL
htmlcomp - UI components
htmlgraph - common graphics functions
htscriptgui - GUI functions registration for Scripter

FMX

fmx.fhtmldraw
fmx.fhttables
fmx.fhtsvg
fmx.fhtclasses
fmx.fhtdefscriptadapter

fmx.htfontawesome
fmx.fhtmlcomp
fmx.fhtmlgraph
fmx.htscriptgui

VCL only units

htdirect2d - Direct2D API
htforms - HTML forms
htgif - GIF support for old Delphi
htmlcatbuttons - TCategoryButtons with HTML support
htmldbcomp - DBlabel and TemplatePanel
htchrometabs - ChromeTabs descendant
htmlsmarttabs - SmartTabs with HTML support
htmlvtree - VirtualTree with HTML support
htremotedebug - remote debugger for Scripter
htscriptdebug - sample debugger for Scripter
htsynpdf - PDF export using SynPDF
htvideo - Video element using libvlc
htzip - ZIP support for Delphi 5-2007.

Canvas classes units

htcanvasdx - Direct2D canvas
htcanvasgdi - GDI canvas
htcanvasgdip - GDI+ canvas
htcanvaslazarus - standard Lazarus canvas
htcanvasopengl - OpenGL canvas
htcanvasskia - Skia canvas
htcanvastex - plain text canvas
fmx.htcanvasFMX - standard FMX canvas
fmx.htcanvasAndroid - native Android canvas
fmx.htcanvasiOS - native iOS canvas
fmx.htcanvasOSX - native OS X canvas
fmx.hcanvasText - plain text canvas
fmx.OSXPDF - native PDF export for OSX

2 Classes

HTML node classes has the following hierarchy:

THtNode - base node class, htmlpars unit.
THtXMLNode - XML node, htxml unit
THtmlNode - base HTML node class, htmlpars unit
TStyledHTMLNode - node with CSS styles support (Style property), htmlcss unit
TElement - general DOM element class, htmldraw / fmx.fhtmldraw unit
THtDocument - HTML document, htmldraw / fmx.fhtmldraw unit
TTableElement - HTML table, httables / fmx.fhttables unit
TTableCell - HTML table cell, httables / fmx.fhttables unit
TTableRow HTML table row, httables / fmx.fhttables unit
TSVGElement - SVG element, htsvg / fmx.fhtsvg unit
TSVGBaseElement - base class for SVG elements, htsvg / fmx.fhtsvg unit
TSVGPathElement
TSVGGElement
TSVGDefsElement
TSVGRectElement
TSVGLineElement
TSVGPolylineElement
TSVGEllipseElement
TSVGCircleElement
TSVGTextElement
TSVGImageElement
TSVGUseElement
TSVGForeignObject
TSVGClipPathElement
TImageElement - HTML image, htmldraw / fmx.fhtmldraw unit
TIframeElement - iframe element, htmldraw / fmx.fhtmldraw unit
TTextElement - HTML text element, htmldraw / fmx.fhtmldraw unit
TDetailsElement - HTML details element, htmldraw / fmx.fhtmldraw unit
TBaseInputElement - basic HTML input element, htmldraw / fmx.fhtmldraw unit
TInputElement - input element - edit, radio, checkbox. htclasses / fmx.fhtclasses unit
TSelectElement - select element. htclasses / fmx.fhtclasses unit

TTextArea element - textarea element. htclasses / fmx.fhtclasses unit

TControlElement -container for VCL/FMX controls, htmldraw / fmx.fhtmldraw unit

TFieldSetElement - fieldset, htclasses / fmx.fhtclasses unit

THtLabelElement - label element, htclasses / fmx.fhtclasses unit

THtCommentElement - comment (!) element, htclasses / fmx.fhtclasses unit

TFrameSetElement - frameset element, htclasses / fmx.fhtclasses unit

TFrameElement - frame element, htclasses / fmx.fhtclasses unit

3 Custom element classes

To register or override element class use HtRegisterElementClass from htmldraw unit

```
procedure HtRegisterElementClass(const Tag: string; const ElementClass: THtElementClas
```

4 Styles

There are several places where CSS styles can be defined

- inline element style = ".." attribute
- document <style> tags
- control Styles property (f.e. HtPanel.Styles)
- HtGlobal.Styles property
- HTML4Stylesheet

When calculating element style, stylesheets from above list are applied in reverse order.

1. Inline element style. F.e. . To check inline style use Element.SelfStyleSheet property.
2. Properties set by HTML4 attributes. F.e. <image width="300"> will add width:300px; to SelfStylesheet.
3. Properties set by HTML4 default stylesheet. F.e. <b> tag will cause font-weight:bold added to calculated style.
4. Properties inherited from parent elements. F.e. direction:rtl can be inherited from parent.
5. Properties calculated from document <style>..</style> section.
6. Properties calculated from Editor.Styles (or HtPanel.Styles)

To add global CSS rules use **HtGlobal.Styles.Add**.
To add new rules to document use **Doc.Styles.Parse(NewCSS)**

By default same stylesheets are cached and same styles are shared between documents.
To prevent this set THTDocument.SharedStyles to false.

5 Quirks mode

Handling of some CSS properties depends on document header (see quirks mode https://developer.mozilla.org/en-US/docs/Web/HTML/Quirks_Mode_and_Standards_Mode)
THtDocument and THtPanel has QuirksModeSelector property which defines whan mode will be choosen

qmAuto - depents on header
qmDocType - treat all documents as having DOCTYPE
qmQuirks - treat all documents as not having DOCTYPE

6 Controls

Most of HTML enabled controls have two main properties: **HTML** for defining HTML content and **Styles** for CSS styles.

VCL controls (htmlcomp unit)

THtHint - HTML enabled hint window
THtBalloonHint - HTML enabled balloon hint window
THtLabel - HTML enabled label
THtCheckBox - HTML enabled checkbox
THtRadioButton - HTML enabled radio
THtListbox - Listbox with HTML enabled items
THtButton - HTML enabled button
THtSpeedButton - HTML enabled speedbutton
THtPopupMenu - HTML enabled popup menu
THtComboList - HTML enabled combo
THtColorCombo - color picker combo
THtTabset - HTML enabled tabset
THtStatusBar - HTML enabled status bar
THtNotiyWindow - HTML enabled notification window
THtMetroPanel - Metro-like panel
THtFloatForm - floating form with HtPanel
THtVirtualTreeView - HTML enabled VirtualTreeView descendant
THtVirtualXMLTree - HTML enabled intercative VirtualTreeView descendant
THtChromeTabs - HTML enabled ChromeTabs
THtSmartTabs - HTML enabled rkSmartTabs
THtCatButtons - HTML enabled category buttons

6.1 HtPanel

THtPanel
Main HTML control, supports scrolling, zoom, gestures, lazy and delayed image loading, text selection, autoheight.

Published Properties:
Active : boolean - enable or disable mouse event processing. Disable for faster scrolling and resize.
AdaptiveZoom: boolean - switch between proportional zoom and browser-like (fit to widow) mode.
AllowFocus : boolean - allow/disallow focus.
AllowScaling : boolean - allow zoom using Ctrl+Wheel.
AutoHeight : boolean - automatic height calculation.

BackgroundImageLoading: boolean - load images in separate thread.
ClearHTMLAftrerLoad: boolean - clear HTML property after document is loaded to reduce memory consumption
CurrentFile : string - full name of file loaded to panel, used to expand path of images referenced by document.
EnableSelection: boolean - enable/disable text selection
HorizontalScrollBar - horizontal scrollbar mode
HTML : TStrings - set HTML content
Images : TImageList - linked Image list. Images from this list can be referenced by index: src="1"
LazyImageLoading: boolean - load image only when it is scrolled into view
HighlightTextColor - color of highlighted text
PreservePositionOnHTMLChange - do not reset scroll offsets when HTML is changed.
QuircksModeSelector - select how to handledocuments without DOCTYPE, see Quicks mode section.
Scale : integer - zoom in percents
ScaleFromCenter: boolean - switch between scaling from left top and mouse cursor
ScaleMin : integer - minimum scale value
ScaleMax : integer - maximim scale value
Script : TStrings - additional pascal scripts
SelectHandleStyle - style of selection handles (on touch-enabled devices)
ShowHing - show hints in elements (title attribute)
Styles : TStrings - additional CSS styles
TouchScroll : boolean - enable scroll by pressing and moving mouse.
VerticalScrollBar - vertical scrollbar mode
WebLoading - enable loading images and styles from web.

Public properties:
Doc : THtDocument - HTML document.
Document : THtDocument - HTML document - same as Doc
CanvasClass - canvas class used for displaying document
ScriptAdapter - document script adapter (see Scripter manual)

Events
AfterControlCreated - called after creating Delphi control
AfterPaint - called after paint (after EndScene)
AfterTableColumnMoved - called after HTML table column was moved
OnGetParam - get parameter for templates in comments (THtTemplatePanel)
OnGetURL - custom handling of CSS stylesheets, font and image URLs.
OnGetImage - custom handling of image URLs
OnURLCLick - called when <a> element is clicked. Use Sender['href'] to get URL.
OnURLEnter - called when mouse enters <a> element. Use Sender['href'] to get URL.

OnURLExit - called when mouse leaves <a> element. Use Sender['href'] to get URL.
OnElementEnter - called when mouse enter any element
OnElementExit - called when mouse leaves any element
OnMouseWheel - called on mouse wheel
OnAnimationEnd - called after animation is finished
OnElementDragEnd - called when element is dropped
OnElementResizeEnd - called when element is resized
OnElementClick - called when element is clicked
OnElementClickEx - same as OnElementClick but with additional parameters
OnAfterImageLoaded - called after image load (lazy or background mode)
OnImageLoadFailed - called if image load was failed
OnScrollBarPaint - paint event for vertical scrollbar
OnCreateControl - custom handling of Delphi controls creation
OnShowResizeHint - custom resize hint handling
OnScroll - called after scroll

Methods
Scroll(DX, DY) - scroll content
ScrollIntoView(Element) - scroll element into view
ScrollIntoCenter - scroll element into window center
ScrollIntoTop - scroll element to window top
ScrollRectIntoView - scroll document rect into view
ScrollCanvasRectIntoView - scroll canvas rectangle into view
CopySelectiontoClipboard - copy selected content to clipboard
CopytoClipboard - copy whole document to clipboard
LoadfromFile - load document from file
LoadfromURL - load document from URL
LoadfromString - load document from string
LoadfromResource - load document from resource
Print - print document
FindDialog - execute find dialog
ReplaceDialog - execute replace dialog
ShowFloatHint - show hint for element
HideFloatHint - hide hint
ScalefromPoint - scale document from point
SetTargetPosition - set target position for smooth scrolling
SetTargetScale - set target scale for smooth zoom

6.2 VirtualTrees

THtVirtualTree and **THtVirtualXMLTree** classes are **TVirtualStringTree** descendants with HTML support.
They are located in a **htmlvtree** unit.

Please note that **htmlvtree_*** package is not installed automatically, you can install it manually after installing VirtualTrees library from GetIt or other source.

7 CSS properties and classes

Properties

Element CSS properties can be accessed via **Element.Style** property. For example

```
function BorderTopStyle: TLineStyle;
function BorderRightStyle: TLineStyle;
function BorderBottomStyle: TLineStyle;
function BorderLeftStyle: TLineStyle;
function BoxShadow: TElementShadow;
function TextShadow: TElementShadow;
function FlexDirection: TCSSFlexDirection;
function JustifyContent: TCSSJustifyContent;
```

Some properties are of **TMeasure** type, to get actual pixel value use its **PIxels** (integer) or **FPixels** (float) property.
Also it has **Value** and **MUnit** (measurement unit) properties.

Properties cannot be changed directly, to change it use Element.AddCSS method. For example

```
Element.AddCSS('color: black');
```

AddCSS has two additional parameters
- **UpdateStyleAttr** - element style attribute will be updated
- **Weight** - custom CSS weight (priority)

To change porperty without affecting style attribute, it is better to use **AddvirtualCSS** method.
To clear CSS property value (revert to default) use **ClearCSS** method, or **ClearCSSRecursive** (clear in element and all descendants).

Property can be changed for number of selected elements, use **css[]** property of **JQuery** or **XPath** result. For example

```
Doc.JQuery('img.small').css['border'] := 'solid red 1px';
```

Inline element stylesheet can be accessed via **SelfStylesheet** property. Note that it contains properties from both inline style and virtual style.
Virtual style contains properties calculated from element attributes and added from code using **AddVirtualCSS**.

To change whole inline style use **SelfStyle** property, **SetAttribute** method or **Attr['style']** property. Attr[] will only change attribute, but not affect stylesheet, **SetAttribute** and **SelfStyle** will immediately apply new value to inline stylesheet.

Classes

- StyleClass or ClassName or Attr['class'] - get current classes
- HasClass - check is element has class
- RemoveClass - remove class from element
- ToggleClass - add or remove class

When using JQuery or XPath following methods are available:

- HasClass - check is element has class
- RemoveClass - remove class from element
- ToggleClass - add or remove class

Example:

```
Doc.JQuery('div.row').addClass('selected');
```

7.1 Listing element properties

Example of getting all CSS properties for element

```
function TForm1.GetCurrentCSS(P: TElement): string;
var WL: TCSSWeightList;
    P: TElement;
    i: integer;
    s: string;
begin
  WL := TCSSWeightList.Create(16);
  Result := '';
  while Assigned(P) do
  begin
    WL.Clear;
    HTML4StyleSheet.GetElementSelectors(P, WL, HTML4StyleSheet.Main,
  [], cmAll, P.Document.ClientWidth, P.Document.DeviceWidth);
    P.Document.Styles.GetElementSelectors(P, WL, P.Documnt.Styles.Main,
  [], cmAll, P.Document.ClientWidth, P.Document.DeviceWidth);
    { inline style }
    if Assigned(P.SelfStyleSheet) then
      P.SelfStyleSheet.GetElementSelectors(P, WL, P.SelfStyleSheet.Main,
        [], cmAll, P.Document.ClientWidth, P.Document.DeviceWidth);
    for i := 0 to WL.Count - 1 do
    begin
      s := CSSValuetoString(WL[i]^);
      Result := '<div class="prop"><span class="propname">' +
        copy(s, 1, FindChar(':', s))  + '</span><span class="propvalue">' +
        copy(s, FindChar(':', s) + 1, MaxInt) +'</span></div>' + Result;
    end;
```

```
    P := P.Parent;
  end;
  WL.Free;
end;
```

8 Unicode

Library supports unicode for non-unicode Delphi versions (5 - 2007).
To enable unicode support, uncomment **{$DEFINE WIDE}** directive in **htmlinc.inc** file and install **TNTUnicode** package.

In WIDE mode hstring is mapped to WideString and hchar is mapped to widechar type.

9 Images

Image handling

Most of HTML components and THtDocument has OnGetImage event.

```
function(Sender: THtDocument; const Url: string; Element: TElement): THtBytes of obje
```

function should be thread-safe and return image data in TBytes or return nil if image cannot be loaded.
Embedded images, encoded using data URI, are processed internally, OnGetImage is not called.
When OnGetImage is not defined or return nil, THtDocument treats URL as file name and tries to load image from file.
In case URL contains only name without path, it is combined with CurrentFile property or application .exe path .

THtPanel has additonal default processing:

When URL is integer number and image list is assigned to Images property, URL is treated as index in image list.
When URL starts with **'http'** or **'https'**, it is treated as web address.
When URL starts with '**/form/**' rest is treated as component name for current form. It can be **TImage** or **TImageList** and additional index, f.e. **/form/ImageList1/2**

Virtual image sources

For defining application-wide image sources, there is a THtVirtualImageSource class

```
THtVirtualImageSource = class
public
  function GetImage(const URL: string; var ImageData: THtBytes; DPI: integer = 96):
end;
```

Each image source is registered with unique prefix, and can be used from any document.
Example:

```
HtGlobal.RegisterVirtualImageSource('_resource', THtVirtualImageRes.Create);
```

There are following predefined sources / prefixes:

- **_resource** - load from resource
- **_shellsmallicons** - shell small icon

- **_shelllargeicons** - shell large icons
- **_dialogicons** - dalog icons
- **_forms** - application forms

Example of image URL:

```
<img src="_dialogicons/information"/>
```

Image lists

Some HTML components has Images property, When image list is assigned, images can be referenced by index.

Lazy loading

THtPanel has LazyLoading and BackgroundLoading properties.
In LazyLoading mode, images are loaded only when they are scrolled into view.
BackgroundImageLoading enables image loading in a separate thread.

Embedding images

To embed all images into document (using data URI) call

```
procedure THtDocument.EmbedAllImages;
```

Encoding and decoding

htimage and fmx.htimage units contains image converter implementation classes for image encoding and decoding.

```
  IHtImageConverter = interface
    function EMFtoSVG(Sender: TObject; var PictData: TBytes; Width, Height: integer; c
    procedure MakeTransparent(Sender:  TObject;var Data: TBytes; var Ext: string; Tra
    function DecodeJPEG(const Data: THtBytes; var Res: THtBitmapData): boolean;
    function DecodeJPEG2000(var Data: THtBytes): boolean;
    function DecodePNG(const Data: THtBytes; var Res: THtBitmapData): boolean;
    function DecodeGIF(const Data: THtBytes; var Res: THtBitmapData): boolean;
    function EncodePNG(ABitmap: THtBitmapData; out Res: TBytes): boolean;
    function EncodeJPEG(ABitmap: THtBitmapData; out Res: TBytes): boolean;
    function HTMLtoPNG(Sender: TObject; const HTML: string; MaxWidth, MaxHeight, PageW
      OnGetImage: THtGetImageData; out Res: TBytes; PagedMedia: boolean): boolean;
    function WMZtoSVG(Sender: TObject; var PictData: TBytes; Width, Height: integer; c
    function SVGtoEMF(Sender: TObject; const SVG: string; var Width, Height: integer)
    function HTMLtoSVG(const HTML, Style: hstring; Width, Height: integer; Paged: bool
  end;
```

To get current image converter implementation use **THtDocumentConverter.ImageConverter** property

10 Canvases

HTML rendering is abstracted from graphics libraries via THtCanvas class. There are several implementations that can be selected globally or for certain component or document. Global variables (defined in htcanvas):

- **HtDefaultCanvasClass** - default canvas for VCL
- **HtUIDefaultCanvasClass** - default canvas for VCL UI controls (Checkbox, Radio)
- **HtFMXDefaultCanvasClass** - default class for FMX

THtDocument and **THtPanel** has **CanvasClass** properties to set preferred canvas.

There are the following canvas implementations:

VCL

THtCanvasDX - rendering using DIrect2D library, best performance and quality, recommended as default class on Windows. May not work on XP. htcanvadx unit.
THtCanvasGP - rendeding using GDI+ library. Good quality but bot so fast as DX. htcanvasgdip unit.
THtCanvasGDI - rendering using GDI. Do not support antialiasing and has integer coordinates. htcanvasgdi unit.
THtCanvasSKIA - good quality and speed, but not so fast as DX. htcanvasskia unit.
THtCanvasText - plain text canvas. htcanvstext unit.
THtCanvasOpenGL - OpenGL canvas, good for displaying large vector data. Work only in window, do use as default canvas.

FMX

THtCanvasFMX - default FMX canvas. On windows may use D2D directly for better speed and more features. fmx.htcanvasfmx unit.
THtCanvasOSX - native OSX canvas. Faster than default. fmx.htcanvasOSX unit. Note that it cannot be used when **GlobalUseMetal** is set to true
THtCanvasAndroid - native Android canvas, faster than default and better quality. fmx.htcamvasAndroid unit.
THtCanvasiOS - native iOS canvas. faster than default and better quality. fmx.htcanvasiOS unit.

To access canvas from document, use **Document.Surface** property.

11 Navigation

DOM Element methods

Count: integer - number of children
Elements[Index] - child element by index
Parent - parent element (nil for root)
Next - next sibling element
NextElement - next element in DOM tree
NextVisibleElement - next visible in a DOM tree
Previous - previous sibling
PreviousElement - previous in a DOM tree
NextLineElement - next line element - text or image element with position = static and float = none.
PreviousLineElement - previous line element - text or image element with position = static and flow = none.
NextNonEmptySibling - next non-empty sibling element. Empty elements are BR and text elements with blank or empty content.
PreviousNonEmptySibling - previous non-empty sibling element Empty elements are BR and text elements with blank or empty content.
Last - last node in tree
FirstChildTag - first non-text child node
LastChildTag - last non-text child node
LastChild - last child node
ChildIndex - Index of element among sibling elements with non-empty tag
HasAsParent(const E: THtNode): boolean;
Root - root node
NodebyName - searches for node in child nodes. Supports paths (Node1/Node2/Node3...)
FindNode - recursive search of node by name (tag)
NodebyAttr - search for node by attribute name and value
NodebyNameAttr - search for node by node tag and attribute
FindNodebyAttr - recursive searching of node by attribute
NodebyNameIndex - node by tag and index

12 JQuery and XPath

THtDocument supports DOM queries using XPath and JQuery. Example:

```
for E in Document.JQuery('img') do ..
```

Returned list has the following properties and methods:

```
function GetCount: integer - count of nodes
function AddClass(const ClassName: string): IHtNodeList - add class
function RemoveClass(const ClassName: string): IHtNodeList - remove class
function ToggleClass(const ClassName: string): IHtNodeList; - toggle class
procedure Remove - delete nodes
property Nodes[Index: integer]: THtNode - node by index
property Attr[const Name: hstring]: hstring  - attributes
property html: hstring - Inner HTML
property css[const Name: hstring]: hstring -  CSS propeties
property Text: hstring - add or get node text
```

Calls can be chained, i.e.

```
JQuery('div').AddClass('some').CSS['color'] := 'green'];
```

13 DOM

Each element in a DOM tree have the following properties

Parent - parent node or nil
Count - number of child elements
Elements[index] - child elements
Attributes[index] - attributes
Attributes.Count - number of attributes (check Attributes for nil!)

Adding new elements

```
function Add(const E: TElement): TElement
      Add E to child elements
function InsertAfter(const EAfter, E: TElement): TElement
      Insert E after EAfter element. EAfter should be child of current element.
function InsertBefore(const EBefore, E: TElement): TElement
      Insert E before EBefore element. EBefore should be child of current element.
function AppendChild(const E: TElement): TElement
      Append E to child elements
function AddHTML(const HTML: hstring): THtmlNode
      Add HTML
function InsertHTML(const HTML: hstring): THtmlNode -
      Insert HTML at top
```

Deleting element

```
procedure DeleteChild(const E: THtNode);
```

Updating elements

To change whole element content use **InnerHtml** and **OuterHTML** properties.

14 Events

DOM elements supports following script events

ondragstart - start drag
ondragend - end drag
ondragenter - executed on target
ondragleave - executed on target
ondrop - executed on drop target
onmousedown - mouse button pressed
onmouseup - mouse button released
onclick - element is clicked
ondblclick - element is doubleclicked
ontransitionent - end of CSS transition / animation
onmouseover - mouse enter
onmouseout - mouse leave
onmousemove - mouse moved
onresize - element is resized
onresizeend - element was resized
onscroll - element content was scrolled

Example:

```
<div onclick="form.MyDivClicked(this)">
```

For more details please refer to Scripter manual.

THtDocument events

BeforePaint - before document paint
OnGetUrl - custom loading of images and stylesheets
OnGetParam - get template parameter
OnGetImage - custom loading of images
OnRepaint - on repaint
OnClick - element is clicked
OnURLEnter - mouse enter <a> element
OnURLExit - mouse leaves <a> element
OnElementEnter - mouse enter element
OnElementExit - mouse leave element
OnSizeChanged - content bounds was changed after layout calculation
OnElementDragEnd - drag end
OnElementDragMoved - drag element moved
OnElementResizeEnd - resize end

OnNewPage - new page (when generating paged layout)
AfterTableColumnMoved - table column was moved
OnSpellCheck - spell checking
OnCreateControl - custom Delphi control creation
AfterControlCreated - after Delphi control was created
OnAnimationEnd - animation end
OnElementDeleted - DOM element deleted
AfterImageLoaded - image was loaded (lazy or background mode)
OnImageLoadedFailed - image loading was failed
BeforeBeginScene - before BeginScene when painting
BeforeEndScene - before end scene when painting
OnShowResizeHint - show custom resize hint
BeforeElementDraw - before element is painted

For HtPanel events please refer to HtPanel 12 section.

Adding events at runtime

```
Element.setAttribute('on' + eventname, EventProc)
```

15 Scripts

THtDocument has language neutral scripts support. Scripts are executed via adapter class

```
THtScriptAdapter = class
public
  constructor Create; virtual;
  procedure AddScript(const Script, ScriptType: hstring); virtual;
  procedure Compile; virtual;
  procedure Clear; virtual;
  procedure OnLoad(const ADocument: THtDocument); virtual;
  procedure Run(Element: TElement; const Event: string; const Script: hstring); virt
  procedure RegisterObject(const Name: string; Value: TObject); virtual;
  function RunFunction(const Element: TElement; Func: TObject; const Params: array o
end;
```

Scripts can be used in element events, f.e. onclick, and in <sscript> document section.
Default implementation supports Pascal scripts, to use it add htdefscripter unit (for VCL) or fmx.fhtdefscriptadapter to uses list.
Note that GUI functions requires separate **htscriptgui** and **fmx.htscriptgui** units.
For more details please refer to Scripter manual.

16 HTML and Plain text

THtDocument and TElement has properties for getting and setting inner and outer HTML and inner text:

InnerHTML - get or set inner HTML
OuterHTML - get or set outer HTML
InnerText - get or set inner text (plain).

Example:

```
E := Doc.GetElementById('mydiv');
if Assigned(E) then
  E.InnerHTML := '<b>New Content</b>';
```

Also HTML can be set using JQuery or XPath. Example:

```
Doc.JQuery('.myclass').html := 'New content';

Adding text to nodes:

Doc.JQuery('.myclass').text := 'Added text content';
```

16.1 Formatted HTML

To get formatted HTML use **THtDocument.FormattedHTML** function.

16.2 Incorrect markup

Library can handle most of incorrect markup cases (missed close tags, quotes, incorrect nested tags, etc.).
When getting HTML back (using OuterHTML) it always be well-formed.

17 XML and JSON

htxml unit contains **THtNode** descendant **THtXMLNode** for parsing XML and JSON. Use

```
constructor Create(const XML: hstring); reintroduce; overload; virtual;
constructor CreateFromFile(const FileName: hstring; Encoding: THtmlEncoding = heDe
constructor CreatefromJSON(const JSON: hstring; dummy: integer = 0); overload;
```

it also contains SAX XML parser class: **THtSAXXMLParser**

18 HTML encode

To encode text to HTML (escape) use the following functions from htmlpars unit:

```
function HtmlEncode(const s: hstring): hstring;
function HtmlEncodeAttr(const s: hstring): hstring;
```

19 Hyphenation

To define hyphenation rules obtain language using

```
THtLanguage.GetLanguage(const LangCode: string): THtLanguage
```

and define rules using

```
procedure AddHyphenation(const Word: hstring; const APositions: TCardinalArray);
procedure LoadHyphenationfromStream(const ST: TStream);
```

20 Delphi controls

Any Delphi control can be embedded to HTML using <control> tag.
First, its should be registered using RegisterClasses procedure, i.e.

```
RegisterClasses([TMemo])
```

Using in HTML:

```
<control type="TMemo"></control>
```

propertied can be set using control tag attributes or Delphi object notation (used in DFM):

```
<control type="TMemo">
  object
  ....
  end
</control>
```

21 Fonts and FontAwesome

Custom fonts can be embedded into document in TTF/OpenType format. When using Office library, WOFF format is also supported.
In addition, custom fonts can be registered directly using canvas class methods

RegisterCustomFont
RegisterCustomFontData

By default documents use shared font collection. To use own font collection (f.e. document is used in thread) pass true to AThreaded parameter of constructor.

FontAwesome

Library has built-in FontAwesome support. To enable it add htfontawesome (for VCL) or fmx.htfontawesome (for FMX) into uses list.
After that you can use standard FA syntax like

```
<span class="fa fa-solid fa-file">
```

22 Hints

Library supports three types if hints

Standatd hints

Set HtPanel.ShowHint to true and use title attribute to set element hint, i.e.

```
<div title="test">
```

HTML hints

Add hthints unit to uses list and use data-hint attributes, f.e.

```
<span data-hint="Test">
```

Additinal hint classes:

.hint--top
.hint--top-left
.hint--top-right
.hint--bottom-left
.hint--bottom-right
.hint--bottom
.hint--right
.hint--left
.hint--small
.hint--medium
.hint--large
.hint--error
.hint--warning
.hint--info
.hint--success
.hint--always
.hint--rounded
.hint--no-animate
.hint--no-shadow

Float hints

Float hints are displayed using float window and can contain large HTML data.
To use it set floathint attribute.

23 Notifications

Functions for displaying notifications (htmlcomp unit)

```
procedure HtNotify(const HTML, AStyles: string);
procedure HtNotifyError(const HTML : string);
procedure HtNotifyInfo(const HTML : string);
procedure HtNotifyWarning(const HTML : string);
```

24 Colors and themes

Following functions are used for color encoding/decoding

```
  function htmlHextoColor(const s: string): cardinal;
  Convert color string in any format to color.

  function htmlHextoColorSys(const s: string; out SysColorFlag: integer): cardinal;
  Convert color string in any format to color, returns System flag for system colors.

  function HtStringtoSysColor(const Color: string): cardinal;
  Convert system color name to color using current theme.

  function htmlColortoHex(C: cardinal): string;
  Convert color to rrbbgg format

  function htmlColortoStr(C: cardinal): string;
  Convert color to #rrbbgg format

  function HtStringtoColor(const Color: hstring): cardinal;
  Convert string color code (f.e. green) to color value. When color not found returns
```

Note, that when using alpha, it should be in last two symbols: #rrggbbaa

There are number or system color names mapped to current system or Delphi theme colors

```
ACTIVEBORDER
ACTIVECAPTION
APPWORKSPACE
BACKGROUND
BTNFACE
BTNHIGHLIGHT
BTNSHADOW
BTNTEXT
BUTTONFACE
BUTTONTEXT
C3DDKSHADOW
C3DFACE
C3DHIGHLIGHT
C3DHILIGHT
C3DLIGHT
C3DSHADOW
CANVAS
CANVASTEXT
CAPTIONTEXT
DESKTOP
FIELD
FIELDTEXT
GRADIENTACTIVECAPTION
GRADIENTINACTIVECAPTION
GRAYTEXT
HIGHLIGHT
HIGHLIGHTTEXT
HOTLIGHT
INACTIVEBORDER
```

```
INACTIVECAPTION
INACTIVECAPTIONTEXT
INFOBK
INFOTEXT
MENU
MENUBAR
MENUHIGHLIGHT
MENUTEXT
SCROLLBAR
WINDOW
WINDOWFRAME
WINDOWTEXT

To define custom color mapping, write own

function ThemeColorProc(ColorID: integer): cardinal;

and register it using

 THtmlNode.SetThemeColorPrc(ThemeColorProc);
```

25 Highlighted text

To highlight text in **HtDocument** set **HighlightText** property.
Highlight color can be set using **HighlightTextColor** property.

26 Document bounds

To calculate HTML bounds call

```
Doc.CalcSize(DesiredWidth)
```

and use **Doc.DocumentWidth** and **DocumentHeight** properties

27 Inputs and forms

In addition to standard HTML edit, <input> element support following types:

date - date edit
time - time edit
number - number edit. Additional attributes: valuetype="float", decimaldigits, maxvalue, minvalue

28 Scale and DPI

THtDocument

To set document scale use **Document.Surface.ScaleFactor** (single). 1 = 100%.
To set current DPI use **Document.Surface.DPI** property. Note that **ScaleFactor** contains final scaling, so for DPI 192 and content scale 200% it should be 4.

HtPanel

```
property Scale: integer - scaling in percents
property ScaleMin: single  - min scale value
property ScaleMax: single - max scale value
property ScaleFromCenter: boolean - scale from mouse cursor or left top.
```

To set target scaling for smooth zoom use

```
procedure SetTargetScale(TargetScale: integer);
```

29 Selection

THtDocument has **Selection** property with the following members:

```
// Position in element text
StartPos, EndPos: integer;
CellMode, SingleTable: boolean;
//Touch markers points
LeftPoint, RightPoint: TPointF;
function InSelection(const E: TElement): boolean;
function PartiallyInSelection(const E: TElement): boolean;
function Empty: boolean;
function Inverted: boolean;
function FullSelected(const E: TElement): boolean;
function RealStartElement: TElement;
function RealEndElement: TElement;
function RealStartPos: integer;
function RealEndPos: integer;
function SelectedRows: integer;
function SelectedCols: integer;
procedure ShrinktoText;
procedure Clear;
procedure SelectAll;
procedure SaveSelection;
procedure RestoreSelection;
procedure Invert;
  // Element containing all selected elements
function TopElement: TElement;
// Table Cell containing all selected elements
function TopCell: TElement;
// Return selected table when selection is in CellMode and single table is selected
function SelectedTable: TElement;
function AtPoint(x, y: integer; out Left: boolean): boolean;
// Return true in CellMode when E is cell of same table as Start or End element or
//  Used in style changing procedure to determine if style should be applied to thi
function IsTopSelectedCell(const E: TElement): boolean;
function SelectedFontCount: integer;
function FirstCell: TElement;
function LastCell: TElement;
function LastSelected: TElement;
// Return selection length in visible symbols (image is treated as single symbol)</
function VisibleLength: integer;
property AbsoluteStart: integer;
property AbsoluteEnd: integer;
property StartElement: TElement;
property EndElement: TElement;
property AllSelected: boolean;
property Text: hstring;
```

To change selection, set StartElement, StartPos, EndElement, EndPos properties.

TTextElement has the following members:

```
function SelectedText: hstring;
function SelectedHTMLText: hstring;
procedure SelectWordAt(CharPos: integer);
procedure SelectPara;
```

30 Resize and drag

Resize

Block and table cell elements can be resized by user. To enable this add css **resize** property to element style. Possible values

- horizontal
- vertical
- both

Resize events:
onresize - element is resized
onresizeend - element was resized

Drag and drop

To enable drag and drop add CSS draggable property to element style. Possible values:

- false
- true
- auto

Drag events:
ondragstart - start drag
ondragend - end drag
ondragenter - executed on target
ondragleave - executed on target
ondrop - executed on drop target

30.1 Sortable containers

Special predefined .**sortable** class turns block element into items container/list where items can be reordered by mouse or moved between two .soirtable containers.

31 Search and Table of Contents

To search for all occurrences of text in THtDocument use

```
function CreateSearchResult(const s: hstring; AllWords: boolean = true): hstring;
```

To create table of contents use

```
function CreateTableofContents: hstring;
```

32 Printing

From HtPanel: call
procedure Print(PrintDialog: TPrintDialog = nil; const PrintScale: single = 1);

From document: create another document, pass print canvas class to constructor and call Print.
Example:

```
PrintDoc := THtDocument.Create(Doc.CanvasClass.PrintCanvasClass);
try
  PrintDoc.OnGetUrl := Doc.OnGetUrl;
  PrintDoc.OnGetImage := Doc.OnGetImage;
  PrintDoc.CurrentFile := Doc.CurrentFile;
  PrintDoc.Parse(Doc.OuterHTML);
  PrintDoc.GeneratePagesForPrint;
  PrintDoc.Surface.Print(PrintDialog);
finally
  PrintDoc.Free
end;
```

HTML Report Library has ready to use print preview window for VCL and FMX in htPreviewFrame and fmx.htPreviewFrame units.
To open print preview call

```
procedure ShowHtPrintPreview(ADocument: THtDocument; const AStyles: string = '';
  const ACaption: string = 'Print Preview'; AWidth: integer = 1200;  AHeight: intege

procedure ShowHtPrintPreview(const AReport: hstring; const AStyles: string = '';
  const ACaption: string = 'Print Preview'; AWidth: integer = 1200;  AHeight: intege
```

33 PDF export

THtDocument has two class methods for PDF export:

```
class function HTMLtoPDF(const AHTML: hstring; const AStyles: hstring = ''): TBytes;
class procedure HTMLtoPDFFile(const AHTML, FileName: string);
```

If you need to explicitly set some document properties, use the following code:

```
D := THtDocument.Create(HtDefaultCanvasClass.PrintCanvasClass); // or HtFMXDefaultCa
try
  D.Parse(AHTML);
  D.GeneratePagesForPrint;
  D.Surface.SavetoPDF(FileName);
finally
  D.Free
end;
```

There are several PDF export implementations included:

Direct export for all platforms

Add **htoffice** to uses list (requires Office library).

Windows

Using **SynPDF** library: add **htsynpdf** unit to uses list.

Using **SKIA**: add **htcanvasSkia** to uses list and call

```
THtPagedExporter.RegisterExporter('PDF', THtSkiaPDFExport);
```

OSX

Set **THtCanvasOSX** as FMX default canvas class or pass it to document constructor in code above.

Android

Set THtCanvasAndroid as default canvas class or pass it to document constructor in code above.

iOS

Set THtCanvasiOS as default canvas class or pass it to document constructor in code above.

34 SVG export

Normal or paged document can be exported into SVG using **THtSVGCanvas** from **htcanvas** unit.
Draw or print using this canvas and get SVG from Document.Surface.Pages[index] (THtPageSVG).

35 SVG creation

Library contains helper class for creating SVG paths from code (htcanvas unit):

```
THtSVGWriter = class(TFastString)
  class function StrokeStyle(AColor: cardinal; const AWidth: single;
    AStyle: THtPenStyle; const DashArray: TSingleArray = nil): string;
  class function StrokeStyleCSS(AColor: cardinal; const AWidth: single;
    AStyle: THtPenStyle; const DashArray: TSingleArray = nil): string;
  // Add command
  function Com(ACommand: hchar): THtSVGWriter;
  // Path Move to point
  function Move(const X, Y: single): THtSVGWriter;
  // Path Move to relative point
  function MoveRel(const dx, dy: single): THtSVGWriter;
  // Path Arc with radius rx, ry from Angle to Angle + sweep
  function Arc(const Rx, Ry, Angle: single; large, sweep: integer;
    const X, Y: single): THtSVGWriter;
  // Path Line to point
  function LineTo(const X, Y: single): THtSVGWriter;
  // Path Line relatively from current position
  function LineRel(const dx, dy: single): THtSVGWriter;
  // Draw line
  function Line(const x1, y1, x2, y2: single; Stroke: Cardinal;
    const StrokeWidth: single = 0): THtSVGWriter;
  function F(const Value: single): THtSVGWriter;
  // Path vertical line
  function V(const Value: single): THtSVGWriter;
  // Path relative vertical line
  function VRel(const Delta: single): THtSVGWriter;
  // Path horizontal line
  function h(const Value: single): THtSVGWriter;
  // Path relative horizontal line
  function HRel(const Delta: single): THtSVGWriter;
  function FS(const Value: single): THtSVGWriter;
  // Add string
  function s(const Value: string): THtSVGWriter;
  // Add color
  function Color(AColor: Cardinal): THtSVGWriter;
  function Stroke(const APen: THtPen): THtSVGWriter;
  function Stroke(AColor: Cardinal; const AWidth: single = 1)
    : THtSVGWriter;
  // Draw path
  function Path(const APath: string; Fill, Stroke: Cardinal;
    const StrokeWidth: single = 0): THtSVGWriter;
  // Begin path
  function BeginPath(Fill, Stroke: Cardinal; const StrokeWidth: single = 0;
    const Style: string = ''; CrispEdges: boolean = false;
    const Attr: string = ''): THtSVGWriter;
  // End path
  function EndPath: THtSVGWriter;
  // Path Z command (close figure)
  function Z: THtSVGWriter;
  // Draw text
```

```
    function Text(const X, Y: single; const Anchor, Value: hstring;
      const Style: string = ''; const Transform: string = '';
      const Attrs: string = ''): THtSVGWriter;
    // Draw circle
    function Circle(const CX, CY, R: single; Fill, Stroke: Cardinal;
      const StrokeWidth: single = 0): THtSVGWriter;
    function Ellipse(const CX, CY, RX, RY: single; Fill, Stroke: Cardinal;
      const StrokeWidth: single = 0): THtSVGWriter;
    // Draw rectangle
    function Rect(const X, Y, W, H: single; Fill, Stroke: Cardinal;
      const StrokeWidth: single = 0; const Style: string = '';
      const RX: single = 0; const RY: single = 0): THtSVGWriter;
    function StrokePath(const Path: THtPath; const Pen: THtPen): THtSVGWriter;
    property Empty: boolean read GetEmpty;
  end;
```

36 Clipboard

htplatform unit contains platform neutral **THtClipboardClass** with the following methods

```
THtClipboard = class
public
  class procedure SetHTML(const AHTML, AText, ASourceFile: hstring);
  class function HasHTML: boolean;
  class function GetHTML: hstring;
  class function HasText: boolean;
  class function GetText: hstring;
  class function HasImage: boolean;
  class function GetBitmap: THtBytes;
  class function GetImage: THtBytes;
  class procedure SetImage(const AImage: THtImage);
  class function HasRTF: boolean;
  class function GetRTF: hstring;
  class function HasMathML: boolean;
  class function GetMathML: hstring;
  class function HasEMF: boolean;
  class function GetEMF: THtBytes;
end;
```