HTML Office Library

Table of Contents

1 Introduction

The HTML Office Library is designed to work with some of the most popular document formats and convert documents to HTML.Converted document can be displayed using HTML Component library or browser. It provides a uniform access to an entire document and its parts, document resources (fonts, images, etc) and properties (title, table of contents, etc).
Library doesn't depend on any external components (DLLs, OLE, ActiveX, etc) and is cross-platform. It is completely written in Delphi and shipped with full source code.

Following document formats are supported:
- Rich Text Format (RTF)
- MS Word 6-2007 binary format (DOC)
- MS Word XML document (DOCX)
- MS Power Point binary format (PPT)
- MS Power Point XML format (PPTX)
- MS Excel binary format (XLS)
- MS Excel XML format (XLSX)
- MS Excel XML binary format (XLSB)
- Adobe PDF format (PDF)
- Supercalc format (SXC)
- EPUB (electronic books).
- FB2 (electronic book}
- Markdown (MD)
- Outlook message format (.eml)
- MIME message (.msg)
- Compiled help (CHM)

Besides the document conversion classes it also contains the following:
- EMF/WMF to SVG conversion
- TTF to WOFF conversion
- TTF normalization
- TTF to SVG conversion
- CFF to TTF conversion
- Adobe PostScript to TTF conversion.
- JPEG200 images
- PICT images

Also library contains classes for accessing the following container files:
- Outlook PST and OST databases.
- The Bat TBB message folders
- ZIP archives
- RAR archives
- Thunderbird message base (MBOX)

Library classes are optimized for fast extracting of plain text content (useful for indexing) without loading images, fonts and other data.

Supported Delphi versions

Delphi 7 - Delphi 11.3

Supported platforms

- Windows 32/64 VCL and FMX
- MacOS
- Linux
- Android
- iOS

Library units

- htoffice - main unit containing document classes
- htofficemail - email document classes
- htofficecode - code parsers for pascal, C++, Java, etc.
- htfonts - font utils unit - classes for working with TTF, WOFF, OTF font formats and PostScript code.
- htimage - image converter for VCL framework.
- fmx.htimage - image converter for FMX framework.
- htsearch - text search engine
- htrar - RAR archives support
- libdeflate - interface to fast deflate library
- openjpeg - JPEG2000

2 General Classes

Document conversion can be accomplished with the following steps:

1. Determine document format using class methods of THtDocumentConverter class (htcanvas unit):

```
class function GetFormat(const ADocument: TStream): string;
class function GetFileFormat(const AFileName: string): string;
```

2. Get converter class using

```
class function GetConverter(const AFileType: string): IHtDocumentConverter;
```

3. Convert file using

```
IHtDocumentConverter = interface
  function ConvertFile(const FileName: string; OnPasswordRequest: THtPasswordRequestEvent = nil): hstri
  function ConvertStream(const Stream: TStream; OnPasswordRequest: THtPasswordRequestEvent = nil): hstr
  function ConvertStreamtoText(const Stream: TStream; OnPasswordRequest: THtPasswordRequestEvent = nil;
    OnText: THtTextCallback = nil): hstring;
  function DocumentClass: TClass;
end;
```

For more complex document processing, create document instance using class returned by DocumentClass meth

THtDocumentConverter class contains class methods for performing general operations on documents without creating document instance and even without knowing actual file/stream format:

```
class function GetConverter(const AFileFormat: string): IHtDocumentConverter;
class function ConverterCount: integer;
class function Converters(Index: integer): THtDocumentConverterRec;
class function GetStreamFormat(const ADocument: TStream): string;
class function GetFileFormat(const AFileName: string): string;
class function ImageConverter: IHtImageConverter;
class function FiletoHTML(const AFileName: string): hstring;
class function StreamtoHTML(const AStream: TStream; AFormat: string = ''): hstring;
class function FiletoText(const AFileName: string; AOnText: THtTextCallback = nil): hstring;
class function StreamtoText(const AStream: TStream; AOnText: THtTextCallback = nil): hstring;
```

2.1 Office document class

THtOfficeDocument is a base class for documents. It contains the following methods and properties:

```
class function ConvertFile(const FileName: string; OnPasswordRequest: THtPasswordRequestEvent = nil): str
Load document from file and convert to HTML

class function ConvertStream(const Stream: TStream; OnPasswordRequest: THtPasswordRequestEvent = nil): st
Load document from stream and convert to HTML

class function ConvertStreamtoText(const Stream: TStream;
     OnPasswordRequest: THtPasswordRequestEvent = nil; OnText: THtTextCallback = nil): hstring; virtual;
Convert stream to plain text

procedure LoadFromFile(const AFileName: string); virtual;
Load document from file

procedure LoadFromBytes(const ABytes: TBytes);
Load document from TBytes array

procedure LoadfromStream(const Stream: TStream);
Load document from stream and convert to HTML

function GetImageData(const ImageID: string): TBytes;
Get document image by image ID (used when EmbedImages is off)

function AsHTML: string;
Get document HTML without CSS styles

function AsHTMLDocument: hstring;
Get document HTML including styles

function AsThumbnail(var ResPNG: TBytes; AMaxWidth: integer = 0; AMaxHeight: integer = 0): boolean;
Get document thumbnail

function Style: string;
Document CSS style
```

```
procedure SavetoStream(AStream: TStream); virtual;
Save document to stream

procedure SavetoFile(const AFileName: hstring); virtual;
Save document to file

function AsText(AOnText: THtTextCallback = nil): hstring;
Get plain text

property OutString: TFastString
Class containing document HTML

property TOC: array of THtTocRec;
Table of contents

property PartCount: integer
Count of parts (sheets, slides, etc.) in document.

property Parts[Index: integer]: TOfficeObject
Document part by index

property Title: string
Document title.

property Author: hstring;
Document author

property KeyWords: hstring;
Document keywords

property LastModifiedBy: hstring;
Last modified

property Created: TDateTime;

property Modified: TDateTime;

property ImageURL: string
URL added to images in HTML when EmbedImages is off.

property EmbedImages: boolean
Determine if image data will be embedded into HTML in base64 format.

property LoadImages: boolean read GetLoadImages write SetLoadImages;
Allows to skip images (convert text and verctor graphics only).

property LoadFonts: boolean read FLoadFonts write FLoadFonts;
Load embedded fomts

property EmbedFonts: boolean read FEmbedFonts write FEmbedFonts;
Embed fonts into converted document

property TextMode: boolean;
Mode for plain text extraction

 property EditableForm: boolean;
Convert PDF forms as active forms

 property MaxLines: integer;
Stop conversion after N lines.

 property MaxPages: integer;
Stop conversion after N pages.

property OnPasswordRequest: THtPasswordRequestEvent;
Password request callback for protected documents.

property DarkMode: boolean;
Convert styles to dark mode.

function AsHTML: string;
Get converted HTML.
```

How to convert document

1. Create document class instance.
2. Set properties (EnbedImages, ImageURL, etc.)
3. Load document using LoadfromFile, LoadfromBytes or LoadfromStream
4. Use AsHTMLDocument to get HTML of whole document or Style + Parts[index].AsHTML to get HTML of document part.

2.2 Office Binary document

TOfficeBinaryDocument is a general class for binary documents: DOC, PPT, XLS. Binary documents are stored in Compound Document File (CDF) format which is similar to FAT.

```
THtOfficeBinaryDocument = class(THtOfficeDocument)
public
  property DocumentStream: TStream read FDocumentStream;
  property Container: TCDFContainer read FContainer;
end;
```

DocumentStream

Main document stream in CDF container.

Container

Compact Document Format container object.

```
TCDFContainer = class
public
  Header: TCDFHeader;
  Directory: array of TCDFDirEntry;
  constructor Create(const ASourceStream: TStream);
  function StreambyName(const Name: string): TStream;
end;
```

2.3 OpenOffice document

TOODocument is a general class for OpenOffice (XML) documents: DOCX, PPTX, XLSX.

```
TOODocument = class(THtOfficeDocument)
public
  property ZIP: TZipFile read FZip;
  property Theme: THtXMLNode read FTheme;
  property ThemeRels: THtXMLNode read FThemeRels;
  property Document: THtXMLNode read FDoc;
  property Styles: THtXMLNode read FStyles;
  property Relations: THtXMLNode read FRels;
end;
```

ZIP

Zip file object

Theme

Document theme XML

ThemeRels

Document theme relations

Document

Document XML

Styles

Document styles XML

Relations

Document relations

2.4 Image converter

Image converter is used by document converters to decode and encode image data. It should be registered using THtDocumentConverter.RegisterImageConverted class method.

```
IHtImageConverter = interface
    function EMFtoSVG(Sender: TObject; var PictData: TBytes; Width, Height: integer; const AStyle: string
    procedure MakeTransparent(Sender:  TObject;var Data: TBytes; var Ext: string; TransparentColor: cardi
    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, PageWidth, PageHeight: i
      OnGetImage: THtGetImageData; out Res: TBytes; PagedMedia: boolean): boolean;
    function WMZtoSVG(Sender: TObject; var PictData: TBytes; Width, Height: integer; const AStyle: string
    function SVGtoEMF(Sender: TObject; const SVG: string; var Width, Height: integer): TBytes;
    function HTMLtoSVG(const HTML, Style: hstring; Width, Height: integer; Paged: boolean = false): hstri
end;
```

Library contains ImageConverter implementations for VCL and FMX frameworks located in htimage and fmx.htimage units.

2.5 Office Art Shape

TOfficeArtShape represents shape object used by binary Office documents - DOC, PPT and XLS. Shape is a complex object containing bitmaps, vector graphics and child shapes. There are number of predefined shape geometries defined by ShapeType property, also shape can have custom geometry.

During conversion to HTML shape is represented as SVG element.

```
TOfficeArtShape = class
public
  Fsp: TOfficeArtFsp;
  ShapeType: integer;
  Props: TOfficeArtProperties;
  R, GroupCoordinateSystem, Anchor: TRect;
  TextType: TPPTXTextType;
  TextIndex: integer;
  Text, Bullet, TextStyles: string;
  Para, Spans: TOfficeTextRuns;
  Shapes: TOfficeArtShapes;
  Parent: TOfficeArtShape;
  Document: TOfficeObject;
  AlwaysOnTop, IsPhoto: boolean;
  Placeholder: TPlaceholderAtomRec;
  Levels: array of record Count: cardinal; Level: integer end;
  constructor Create(ADocument: TOfficeObject);
  destructor Destroy; override;
  procedure ReadfromStream(AStream: TStream; AEndPos: Longint);
  function ShapeLocation: TRectF;
  function GetShapeName(Id: integer): string;
  procedure AddPara(Start, Count: integer; const Style: TCSSStyleSet; const Bullet: string; Level: integ
  procedure AddSpan(Start, Count: integer; Style: PCSSStyleSet;  Color: cardinal; FontIndex: integer);
  function AsHTML(AObject: TOfficeObject; const AStyle: string; PR: PRect = nil): string;
  function MasterShapeID: integer;
  function ShapeFromLibrary(const ShapeName, Fill, Stroke: string; var TextR: TRect; StrokeStyle: THtPen
  function IsBackground: boolean;
  function IsHidden: boolean;
  function IsUserDrawn: boolean;
  function GroupFlags: cardinal;
  function TextWrap: TWhiteSpace;
end;
```

FSP

Shape flags:

```
TOfficeArtFSP = packed record
  id: cardinal;
  Flags: cardinal;
  function IsGroupShape: boolean;
  function IsChildShape: boolean;
  function IsTopmostGroupShape: boolean;
  function IsDeleted: boolean;
  function IsOLEShape: boolean;
  function HorizontalFlip: boolean;
  function VerticalFlip: boolean;
  function IsConnector: boolean;
  function HaveAnchor: boolean;
  function IsBackground: boolean;
  function HaveTypeProp: boolean;
end;
```

ShapeType

Predefined shape type - rectangle, ellipse, arrow, etc.

Props

Set of shape properties

```
TOfficeArtProperty = packed record
  opid: word;
  Value: integer;
  Data: TBytes;
  Name, SValue: string;
end;
```

R, GroupCoordinateSystem, Anchor

Shape bounds, coorginate system for child shape and shape anchor.

TextType (PPTX only)

Shape text type:

```
TPPTXTextType = (Tx_TYPE_TITLE, Tx_TYPE_BODY, Tx_TYPE_NOTES, Tx_None, Tx_TYPE_OTHER,
   Tx_TYPE_CENTERBODY, Tx_TYPE_CENTERTITLE, Tx_TYPE_HALFBODY, Tx_TYPE_QUARTERBODY);
```

Text, Bullet

Shape text and bullet text

Para, Spans

Paragraphs and spans of shape text

```
TOfficeTextRun = record
  StartPos, Count, Level: integer;
  Bullet: string;
  Style: TCSSStyleSet;
  Color: cardinal;
  FontIndex: integer;
end;
```

Shapes

Child shapes

Parent

Parent shape

Document

Document containing shape

3 MS Word DOC

MS Word binary file format is supported for Word versions from Word 95.
Word document is a sequence of paragraphs and spans. Both has positions in global text block and style ID referencing set of style properties.

```
 TWordDocument = class(THtOfficeBinaryDocument)

 FIB: TFIB;
Word file information block

 Styles: TWordStyleSheet;
Word document styles

 Text: TWordText;
Word text block

 ParaList, SpanList: array of TWordPara;
Document paragraphs and spans

 Fonts: array of TWordFont;
Document fonts

 Footnotes: array of TWordFootnote;
Document footnotes

 TextBoxes: array of TWordTextBox;
Document textboxes

 FloatPics: array of TWordFloatPic;
Document floating pictures

 Shapes: TOfficeArtShapes;
Document shapes

Pictures: array of TOfficeArtPicture;
Document pictures

 ListOverride: array of TWordListOverrideData;
 ListFormat: array of TWordListFormat;
 LVLList: array of TLVLF;
List styles
```

StyleSheets

StyleSheets contains paragraphs and spans style properties.

```
 TWordStyleSheet = record
   STSH: TSTSH;
   Items: array of TWordStyleItem;
   function FindStyle(id: integer): integer;
 end;

 TWordStyleItem = record
   STDF: TSTDF;
   Name: UnicodeString;
   Props: TWordPropertyList;
   function ItemCount: integer;
 end;

 TWordPropertyList = record
   istd: word; //Style ID
   Props: array of TWordproperty;
   function FindSPRM(SPRM: integer): PWordProperty;
   function HasBoolValue(SPRM: integer): boolean;
 end;
```

Paragraphs and Spans

```
TWordPara = record
  StreamStart, StreamEnd: integer;
  Props: TWordPropertyList;
end;
```

Word Text

Word text field **contains** document text

```
  TWordTextBlock = record
    StreamStart, StreamEnd: integer;
    ANSI: boolean;
    s: string;
    Prm: integer;
    function CharCount: integer;
    function BytesinChar: integer;
  end;

  TWordText = record
    Blocks: array of TWordTextBlock;
    function TextRange(AStreamStart, AStreamEnd: integer): string;
    procedure AddBlock(AStremStart, AStreamEnd: integer; const AValue: string; AANSI: boolean; APrm: inte
    function TextLength(AStreamStart, AStreamEnd: integer): integer;
    function TextRangeCP(CPStart, CPEnd: integer): string;
    function StreamPostoCharPos(StreamPos: integer): integer;
  end;
```

Footnotes

Contains document footnotes

```
  TWordFootnote = record
    DocStartPos, DocEndPos: integer;
    Text: string;
  end;
```

Text Boxes

Contains document text boxes

```
  TWordTextBox = record
    DocStartPos, DocEndPos, Anchor: integer;
    id: cardinal;
    Text: string;
    R: TRect;
  end;
```

Floating pictures

Contains floating (in text) pictures

```
  TWordFloatPic = record
    Anchor: integer;
    id: cardinal;
    Wrap: TSPATextWrap;
    Clear: TCSSClears;
    R: TRect;
  end;
```

Shapes

Contains document shapes (see OfficeArtShape section).

```
TOfficeArtShapes = class(THtObjectList)
public
  function FindShape(Id: cardinal): TOfficeArtShape;
  function FindShapeRecursively(Id: cardinal): TOfficeArtShape;
  property Shapes[Index: integer]: TOfficeArtShape read GetShape; default;
end;
```

Pictures
Contains document common pictures

```
TOfficeArtPicture = record
  Data: TBytes;
  PicType: string;
  Width, Height: integer;
end;
```

List styles

Contains list style definitiions

```
TLVLF = packed record
  iStartAt: integer;
  NumberFormat: byte;
  Flags: byte;
  Placeholders: array[0..8] of byte; //Each integer specifies a one-based character offset to a level p
  FollowChar: byte; //An unsigned integer that specifies the character that follows the number text.
  IndentRemove: integer; //):  If fIndentSav is nonzero, this is a signed integer that specifies the si
  Unused: cardinal;
  CharPropSize: byte; //An unsigned integer that specifies the size, in bytes, of the grpprlChpx in the
  PropSize: byte; //An unsigned integer that specifies the size, in bytes, of the grpprlPapx in the LVL
  RestartLim: byte; //An unsigned integer that specifies the first (most-significant) zero-based level
  grfhic: byte;
  NumberText: UnicodeString;
end;
```

4 MS Word DOCX

DOCX is an OpenOffice (XML) document format used by MS Word from version 2007. It contains document content, styles and media files packed into single .zip file.

```
TDOCXDocument = class(TOODocument)
public
  function RelIDtoFileName(const RelID: string): string;
  procedure SaveDOCX(const AFileName: string);
  function AllBookmarks: IHtNodeList;
  function AllSDT: IHtNodeList;
  function AllHeaders: IHtNodeList;
  function AllFooters: IHtNodeList;
  function AllTables: IHtNodeList;
  function AddHeaderFooter(AIsHeader: boolean): TDOCXHeaderFooterNode;
  function CopyHeaderFooter(ASource: TDOCXHeaderFooterNode): string;
  property ContentTypes: THtXMLNode;
  property Subject: string;
  property Numbering: THtXMLNode;
  property HasHeader: boolean;
  property HasFooter: boolean;
end;
```

RelIDtoFileName

Convert relation id to file name using main relations table.

SaveDOCX

Save changed document to DOCX file.

AllBookmarks

List of document bookmarks

AllSDT

List of Structured Document Tags.

AllHeaders

List of page headers.

AllFooters

List of all footers.

AllTables

List of all tables

AddHeaderFooter

Add new header or footer to document.

CopyHeaderFooter

Copy header or footer between documents.

ContentTypes

Document content types table.

Subject

Document subject.

Numbering

XML with numbering styles

HasHeader

Document contains page header

HasFooter

Document contains page footer

Document nodes classes

Basic node class.

```
TDOCXNode = class(THtXMLNode)
public
  class procedure ParseStyle(const PX: THtXMLNode; var Style: TCSSStyleSet; Document: TDOCXDocument);
  procedure AsHTML(AOut: TFastString);
  procedure Delete;
end;
```

ParseStyle

Parse DOCX styles node and set CSS style properties.

AsHTML

Save node to HTML output.

Delete

Delete node from document.

Structured Document Tag class w:sdt

```
TDOCXSDTNode = class(TDOCXNode)
public
  property Id: string;
  property PlaceHolder: string;
  property SDTag: string;
  property Content: TDOCXNode;
  property Text: string;
end;
```

ID

SDT id: w:sdtPr/w:id

PlaceHolder:

SDT placeholder id. w:sdtPr/w:placeholder/w:docPart

SDTag

Tag name: w:sdtPr/w:tag

Content

Content node: w:sdtContent

Paragraph class w:p

```
TDOCXPNode = class(TDOCXNode)
public
  Style: TCSSStyleSet;
  function HTMLTag: string;
  function IsEmpty: boolean;
  function TabOffset(Index: integer): integer;
  function IsNumbered: boolean;
  function Numbering: TDOCXNumbering;
  function AddSpan: TDOCXRNode;
  property ListLevel: integer;
  property ListId: integer;
  property StyleClass: string;
end;
```

Style

CSS style

HTMLTag

div, h1-h6 or li.

IsEmpty

Return true if node contains w:r subnodes.

TabOffset

Tab offset for tab index. Offset is calculated from w:pPr/w:tabs node or as tab index multiplied to 32,

IsNumbered

Return true if paragraph belongs to numbered list.

Numbering

List numbering style.

AddSpan

Add w:r node to paragraph.

ListLevel

List level for numbered paragraph.

ListID

Unique list ID.

StyleClass

Paragraph style id: w:pPr/w:pStyle

Span class w:r

```
TDOCXRNode = class(TDOCXNode)
public
  Style: TCSSStyleSet;
  property Text: string;
  property StyleClass: string;
end;
```

Style

CSS style

Text

Plain text

StyleClass

Span style id: w:rPr/w:rStyle

Alternate content class mc:alternatecontent

```
TDOCXAlternateNode = class(TDOCXNode)
public
  function Choice: TDOCXNode;
  function Fallback: TDOCXNode;
end;
```

Drawing node w:drawing

```
TDOCXDrawingNode = class(TDOCXNode)
public
  function ShapeRect: TRectF;
  function MarginRect: TRectF;
  function Floating: boolean;
  function FloatRight: boolean;
  function FloatBottom: boolean;
  function FixedVerticalPosition: boolean;
  function FixedHorizontalPosition: boolean;
  property TextWrap: TOOTextWrap;
end;
```

ShapeRect

Shape bounds

MarginRect

Shape margins

Floating

Is shape floating or inline.

FloatRight

Shape is floated to right

FloatBottom

Shape is floated to bottom

FixedVerticalPosition

Position is relative to page bounds (not to paragraph bounds)

FixedHorizontalPosition

Position is relative to page bounds (not to paragraph bounds)

TextWrap

Text wrapping in shape

Table node w:tbl

```
TDOCXTableNode = class(TDOCXNode)
public
  function ColumnCount: integer;
  function TableWidth: string;
  property StyleClass: string;
  property Width: string;
  property CellTopMargin: single;
  property CellBottomMargin: single;
  property CellLeftMargin: single;
  property CellRightMargin: single;
  property Columns[Index: integer]: TDOCXNode;
end;
```

ColumnCount

Count of table columns

TableWidth

Table wiidth in px or percents.

StyleClass

Table style id.

Width

Table wiidth in px or percents.

CellTopMargin, CellBottomMargin, CellLeftMargin, CellRightMargin

Cell margins in px.

Columns

Column properties.

Table row node w:tr

```
TDOCXTableRowNode = class(TDOCXNode)
public
  function Height: string;
  function SkippedColumns: integer;
  property Cells[ColIndex: integer]: TDOCXTableCellNode;
end;
```

Height

Row height in px or percents.

SkippedColumns

Cells skipped before first cell

Cells

Cells by index

Table cell node w:tc

```
TDOCXTableCellNode = class(TDOCXNode)
public
  function ColSpan: integer;
  function Column: integer;
  function IsVMergeRestart: boolean;
  function VMerged: boolean;
end;
```

IsVMergedRestart

Vertical merging starts at this cell

VMerged

Cell is vertically merged with previous cell

Word shape node wps:wsp

```
TDOCXWordShapeNode = class(TDOCXNode)
public
  function ShapeRect: TRectF;
  function Rotation: integer;
  function FlipVertical: boolean;
  function FlipHorizontal: boolean;
  function PresetGeometry: string;
  function CustomGeometry: THtXMLNode;
  function BorderStyle: THtStroke;
  function SolidFill: string;
  function GradientFill: string;
  function ImageFill: THtXMLNode;
  function PaddingRect: TRectF;
end;
```

ShapeRect

Shape bounds

Rotation

Shape rotation in degrees

FlipVertical

Shape is flipped vertically

FlipHorizontal

Shape is flipped horizontally

ResetGeometry

Shape geometry from library

CustomGeometry

Shape custom geometry

BorderStyle

Shape border

SolidFill

Shape fill color

GradientFill

Shape gradient fill

ImageFill

Shape image

PaddingRect

Shape paddings

Word picture node pic:pic

```
TDOCXPicNode = class(TDOCXWordShapeNode)
public
  property PicID: string read GetPicID write SetPicID;
  ///<summary>Crop of source image in percents for each side</summary>
  property CropRect: TRectF read GetCropRect write SetCropRect;
  ///<summary>Stretch rectangle in percents for each side</summary>
  property StrectRect: TRectF read GetStretchRect write SetStretchRect;
  property Stretched: boolean read GetStretched write SetStretched;
  property Tiled: boolean read GetTiled write SetTiled;
  property TileScale: TPointF;
end;
```

PicID

Picture ID (relation).

CropRect

Crop of source image in percents for each side

StretchRect

Stretch rectangle in percents for each side

Stretched

Is picture stretched

Tiled

Is picture tiled

TileScale

Scaling for tiles.

Header/Footer node

```
TDOCXHeaderFooterNode = class(TDOCXNode)
public
  function IsHeader: boolean;
  function PageType: THeaderFooterPageType;
  function FileName: string;
  function RelsFileName: string;
  property HeaderFooterDocument: TDOCXNode;
  property Relations: TOORelations;
end;
```

IsHeader

Is node header or footer

PageType

Page type: THeaderFooterPageType = (hfpNone, hfpDefault, hfpFirst, hfpEven, hfpOdd);

FileName

Header/footer content file name

RelsFileName

Header/footer relations file name

HeaderFooterDocument

Header/footer content

Relations

Reader/footer relations document.

Bookmark node w:bookmarkStart

```
TDOCXBookmarkNode = class(TDOCXNode)
public
  function Text: string;
  property Id: string;
  property Name: string;
end;
```

Text

Bookmark plain text

id

Bookmark id:

Name

BookMark name

4.1 Code samples

Set bookmark text

```
procedure TForm1.AddTextBookmarksClick(Sender: TObject);
var D: TDOCXDocument;
    P, X, LastP: THtNode;
begin
  D := TDOCXDocument.Create;
  try
    D.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'Sample.docx');
    for P in D.AllBookmarks do
    begin
      X := P.NextNode;
      LastP := nil;
      while Assigned(X) and (X.Tag <> 'w:bookmarkEnd') and (X.Tag <> 'w:r') do
      begin
        if X.Tag = 'w:p' then
          LastP := X;
        X := X.NextNode;
      end;
      if Assigned(X) and (X.Tag = 'w:r') then
        TDOCXRNode(X).Text := 'New text in bookmark'
      else
      if Assigned(LastP) then
        TDOCXPNode(LastP).AddSpan.Text := 'New text in bookmark'
    end;
    D.SaveDOCX(ExtractFilePath(Paramstr(0)) + 'Samplenew.docx');
  finally
    D.Free;
  end;
end;
```

Get all bookmarks

```
procedure TForm1.BookmarkBtnClick(Sender: TObject);
var D: TDOCXDocument;
    P: THtNode;
begin
  D := TDOCXDocument.Create;
  try
    D.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'Sample.docx');
    for P in D.AllBookmarks do
      Memo1.Lines.Add(TDOCXBookmarkNode(P).Name + ': ' + TDOCXBookmarkNode(P).Text);
  finally
    D.Free;
  end;
end;
```

Copy header from one document to another with all relations

```
procedure TForm1.CopyHeaderBtnClick(Sender: TObject);
var D, D2: TDOCXDocument;
    H: TDOCXHeaderFooterNode;
begin
  D := TDOCXDocument.Create;
  D2 := TDOCXDocument.Create;;
  try
    D.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'Sample.docx');
    D2.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'SomeOtherDoc.docx');
    H := D.AllHeaders[0] as TDOCXHeaderFooterNode;
    D2.CopyHeaderFooter(H);
    D2.SaveDOCX(ExtractFilePath(Paramstr(0)) + 'SomeOtherDoc 2.docx');
  finally
    D.Free;
    D2.Free;
  end;
end;
```

Delete all headers

```
procedure TForm1.DelHeaderBtnClick(Sender: TObject);
var D: TDOCXDocument;
    P: THtNode;
begin
  D := TDOCXDocument.Create;
  try
```

```
    D.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'Sample.docx');
    for P in D.AllHeaders do
      TDOCXHeaderFooterNode(P).Delete;
    D.SaveDOCX(ExtractFilePath(Paramstr(0)) + 'SampleNew.docx');
  finally
    D.Free;
  end;
end;
```

Get all structured tags

```
procedure TForm1.SDTBtnClick(Sender: TObject);
var D: TDOCXDocument;
    P: THtNode;
begin
  D := TDOCXDocument.Create;
  try
    D.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'Sample.docx');
    for P in D.AllSDT do
    if TDOCXSDTNode(P).SDTag <> '' then
    begin
      Memo1.Lines.Add(TDOCXSDTNode(P).SDTag + ': ' + TDOCXSDTNode(P).Text);
      Memo1.Lines.Add('');
    end;
  finally
    D.Free;
  end;
end;
```

Set structured tags text

```
procedure TForm1.SetTagsTextBtnClick(Sender: TObject);
var D: TDOCXDocument;
    P: THtNode;
begin
  D := TDOCXDocument.Create;
  try
    D.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'Sample.docx');
    for P in D.AllSDT do
      if TDOCXSDTNode(P).SDTag <> '' then
         TDOCXSDTNode(P).Text := TDOCXSDTNode(P).SDTag;
    D.SaveDOCX(ExtractFilePath(Paramstr(0)) + 'SampleNew.docx');
  finally
    D.Free;
  end;
end;
```

Remove watermark from header

```
procedure TForm1.WatermarkBtnClick(Sender: TObject);
var D: TDOCXDocument;
    P, X: THtNode;
begin
  D := TDOCXDocument.Create;
  try
    D.LoadFromFile(ExtractFilePath(Paramstr(0)) + 'Sample.docx');
    for P in D.AllHeaders do
    begin
      for X in TDOCXHeaderFooterNode(P).HeaderFooterDocument.XPath('//w:sdt') do
        if TDOCXNode(X).Node['w:sdtPr/w:docPartObj/w:docPartGallery/@w:val'] = 'Watermarks' then
          X.Parent.DeleteChild(X);
    end;
    D.SaveDOCX(ExtractFilePath(Paramstr(0)) + 'SampleNew.docx');
  finally
    D.Free;
  end;
end;
```

5 Powerpoint PPT

PPT is a PowerPoint binary file format. Presentation slides, master slides, pictures and other data is stored in Compound Document File storage.

```
TPowerPointDocument = class(THtOfficeBinaryDocument)
public
  TextInfo: TPPTTextInfoContainer;
  Pictures: array of TOfficeArtPicture;
  function SlideCount: integer;
  property Slides[Index: integer]: TPowerPointSlide;
  property MasterSlides[Index: integer]: TPowerPointSlide;
end;
```

TextInfo

Storage for document fonts and common text styles

```
TPPTTextInfoContainer = record
  Fonts: array of TPPTFontCollectionEntry;
  ParaStyle, SpanStyle: TStyleTypeLevelArray;
  DefaultSpanStyle: TCSSStyleSet;
end;
```

Pictures

Document binary pictures

```
TOfficeArtPicture = record
   Data: TBytes;
   PicType: string;
   Width, Height: integer;
end;
```

Slides, MasterSlides

Provides access to document slides and master slides.

```
TPowerPointSlide = class(TOfficeObject)
 public
  Owner: TPowerPointDocument;
  Shapes: TOfficeArtShapes;
  function IsCollapsed: boolean;
  function HasNonOutlineData: boolean;
  function HasMasterShape(ShapeID: cardinal): boolean;
  function AsHTML: string; override;
end;
```

Shapes

Slide shapes. See Office Art Shape section.

IsCollapsed

Slide is collapsed in a slide tree.

HasNonOutlineData

slide contains data other than text in a placeholder shape

HasMasterShape

Shape with specified ID has parent shape in master slide.

AsHTML

Slide HTML (without styles).

6 Powerpoint PPTX

PPTX is an **OpenOffice** (XML) presentation format used by MS PowerPoint from version 2007. It contains presentation slides, master slides, pictures and other media files packed into single .zip file.
Presentation consists of content slides. Each content slide references slide layout and each slide layout references master slide.
Each slide consists of slide shapes. Shapes can be nested and can refer to shape in slide layout or master slide.

```
TPPTXDocument = class(TOODocument)
public
  SlideWidth, SlideHeight: integer;
  function SlideCount: integer;
  function SlideLayoutCount: integer;
  function MasterSlideCount: integer;
  property Slides[Index: integer]: TPPTXSlide;
  property MasterSlides[Index: integer]: TPPTXSlide;
  property SlideLayouts[Index: integer]: TPPTXSlide;
end;
```

SlideWidth, SlideHeight

Default slide size in pixels.

SlideCount

Number of content slides in presentation

SlideLayoutCount

Nunber of slide layouts

MasterSlideCount

Number of master slides

Slides

Content slide by index

MasterSlides

Master slide by index

SlideLayouts

Slide layout by index

TPPTXSlide class

```
TPPTXSlide = class(TOfficeObject)
public
  Name: string;
  Parent: TPPTXSlide;
  Owner: TPPTXDocument;
  function AsHTML: string; override;
  function Master: TPPTXSlide;
  function ShapeCount: integer;
  property Shapes[Index: integer]: TPPTXShape;
end;
```

Name

Slide title

Parent

Master slide for slide layout or slide layout for content slide.

Owner

Presentation document

AsHTML

Slide converted to HTML

Master

Master slide

ShapeCount

Count of slide shapes

Shapes

Slide shapes by index

TPPTXShape class

```
TPPTXShape = class
public
  Node: THtXMLNode;
  Slide: TPPTXSlide;
  MasterShape: TPPTXShape;
  ShapeType, ShapeID: string;
  function AsHTML(AddText: boolean): string;
  function ShapeCount: integer;
  property Shapes[Index: integer]: TPPTXShape read GetShape;
end;
```

Node

XML Node in slide document

Slide

Slide containing this shape

MasterShape

Reference to shape in slide layout or master slide

ShapeType

Shape type (title, etc)

ShapeID

Unique shape ID

AsHTML

Shape HTML

ShapeCount

Number of child shapes

Shapes

Child shape by index

7 Excel XLS

XLS is an Excel binary format based on a Compound Document Format.

```
TExcelDocument = class(THtOfficeBinaryDocument)
public
  DefColWidth: integer;
  function SheetCount: integer;
  property Sheets[Index: integer]: TExcelSheet;
  property Encrypted: boolean;
end;
```

DefColWidth

Default coolumn width in pixels.

SheetCount

Number of sheets in book

Sheets

Sheet by index

Encrypted

Is book encryped.

TExcelSheet class

```
TExcelSheet = class(TOfficeObject)
public
  DefColWidth: integer;
  Cols: TExcelCols;
  Rows: TExcelRows;
  ShowGrid: boolean;
  function IsEmptyCol(Index: integer): boolean;
  property Cells[Row, Col: integer]: PExcelCell;
end;
```

DefColWidth

Default column width in pixels.

Cols

Sheet columns

```
TExcelCol = record
  WidthPx: single;
  Hidden: boolean;
end;
```

Rows

Ssheet rows

ShowGrid

Add grid lines for all cells when export to HTML

IsEmptyCol

Check if column contains only empty cells

Cells

Cell by column/row index

TExcelCell type

```
TExcelCell = record
  Value, Format, HTMLContent: string;
  HTMLContentCols, HTMLContentRows: integer;
  FloatValue: double;
  Style: TCSSStyleSet;
  Spanned: boolean;
  ColSpan, RowSpan: integer;
end;
```

Value

Cell value converted to string.

Format

Cell format (date format, number format, etc)

FloatValue

Numeric value for cell with numbers

Style

Set of cell style properties

ColSpan, RowSpan

Column and row span values (=0 when cell is not spanned)

HTMLContent

HTML block representing complex object linked to cell (f.e. chart(

HTMLContenCols, HTMLContentRows

Number of columns and rows occupied by object

8 Excel XLSX

XLSX is an **OpenOffice** (XML) spreadsheet format used by MS Excel from version 2007.
XLSX file contains number of sheets, each sheet contains cell values and complex objects (charts, etc.) linked to anchor cell.

```
TXLSXDocument = class(TOODocument)
public
  function SheetCount: integer;
  property ConvertDrawings: boolean;
  property Sheets[Index: integer]: TExcelSheet;
end;
```

SheetCount
Number of sheets in document

ConvertDrawing
Enable embedded drawings conversion

Sheets
Sheet by index

TExcelSheet class

```
TExcelSheet = class(TOfficeObject)
public
  DefColWidth: integer;
  Cols: TExcelCols;
  Rows: TExcelRows;
  ShowGrid: boolean;
  function IsEmptyCol(Index: integer): boolean;
  property Cells[Row, Col: integer]: PExcelCell;
end;
```

DefColWidth

Default column width in pixels.

Cols

Sheet columns

```
TExcelCol = record
  WidthPx: single;
  Hidden: boolean;
end;
```

Rows

Ssheet rows

ShowGrid

Add grid lines for all cells when export to HTML

IsEmptyCol

Check if column contains only empty cells

Cells

Cell by column/row index

TExcelCell type

```
TExcelCell = record
  Value, Format, HTMLContent: string;
  HTMLContentCols, HTMLContentRows: integer;
  FloatValue: double;
  Style: TCSSStyleSet;
  Spanned: boolean;
  ColSpan, RowSpan: integer;
end;
```

Value

Cell value converted to string.

Format

Cell format (date format, number format, etc)

FloatValue

Numeric value for cell with numbers

Style

Set of cell style properties

ColSpan, RowSpan

Column and row span values (=0 when cell is not spanned)

HTMLContent

HTML block representing complex object linked to cell (f.e. chart(

HTMLContenCols, HTMLContentRows

Number of columns and rows occupied by object

9 Adobe PDF

Adobe Portable Document Format is a binary file format containing text, vector graphics and images divided into pages. During reading and conversion to HTML library performs the following operations:

- Text is converted to unicode and combined where possible.
- Image masks are applied to images
- Bitmap images are converted to PNG format.
- Colorspaces are applied to images, text and graphics.
- Embedded fonts of all types (including PostScript fonts) are converted to TTF outlines.
- TTF fonts are normalized using font dictionary.
- All fonts are packed to WOFF.
- Lines used to draw underlined text are detected and converted to normal text underline attributes.
- Links are converted to <a> tag.
- Link destinations get attached to text where possible.
- Styles get indexed and assigned to elements as classes (to minimize document size).

```
TPDFDocument = class(THtOfficeDocument)
public
  Catalog: TPDFDictionary;
  constructor Create; override;
  destructor Destroy; override;
  function Encrypted: boolean;
  function PageCount: integer;
  property Pages[Index: integer]: TPDFPage;
  property Destinations: TPDFDictionary;
  property EncryptionDictionary: TPDFDictionary;
  property InfoDictionary: TPDFDictionary;
  property Fonts: THtObjectDictionary;
  property EditableForm: boolean;
end;
```

Catalog

Document catalog: general document object containing references to other dictionaries - pages, outlines, etc.

Encrypted

Is document encrypted or not.

PageCount

Number of pages

Pages

Page by index

Destinarions

Document destinations dictionary. Destinations are used by links inside document.

EncryptionDictionary

Document encryption dictionary.

InfoDictionary

Document information dictionary: Title, Author, etc.

Fonts

Document fonts.

EditableForm

Make PDF form editable (all fields are converted to **input** tags).

PDF page

```
TPDFPage = class(TOfficeObject)
public
  Page: TPDFDictionary;
  Contents: TPDFObject;
  Width, Height, PageIndex: integer;
  ConData: TBytes;
  constructor Create(AOwner: TPDFDocument; APageObject: TPDFDictionary);
  function Resources: TPDFDictionary;
  property Rotate: string;
  property MediaBox: TRectF;
end;
```

Page

Reference to Page dictionary

Contents

PDF object containing page content

Width, Height

Page size

PageIndex

Page number.

ConData

Page contents

Resources

Page resources dictionary

Rotate

Page rotation (degrees)

MediaBox

Page box.

PDF dictionary

```
TPDFDictionary = class(TPDFObject)
public
  function IntParam(const Name: string; ADefault: integer = 0): integer;
  function FloatParam(const Name: string; ADefault: double = 0): double;
  function BoolParam(const Name: string; ADefault: boolean = false): boolean;
  function ContainsKey(const AName: string): boolean;
  property Strings[const AName: string]: string read GetString; default;
  property Bytes[const AName: string]: TBytes read GetBytes;
  property Objects[const AName: string]: TPDFObject read GetObject;
end;
```

IntParam

Get integer value

FloatParam

Get float value

BoolParam

Get boolean value

ContainsKey

Check if key is present in dictionary

Strings

Get string value

Bytes

Get binary value

Objects

Get object value

9.1 Form fields

Document form fields by default are converted to <p> elements with class="pdfformfield" and name attribute set to field name.
When EditableForm property is set to true, form fields are converted to <input> tags and form can be edited in HtPanel/browser.

10 EPUB

EPUB is an e-book file format containing HTML, images and styles packed into single .ZIP file.

```
TEPUBDocument = class(TOODocument)
public
  procedure Parse; override;
  function GetImageData(const AImageID: string): TBytes; override;
end;
```

During conversion process library embed related styles and images into HTML and add page breaks between chapters.

11 Outlook EML

Outlook message format (.EML) is a container for single email message. Usually it contains message body in RTF format and attachments.

```
TOutlookMessage = class(THtOfficeBinaryDocument)
public
  IsUnicode: boolean;
  Recepients: array of record Name, Address: hstring end;
  HTMLBody: hstring;
  Subject, FromName, FromAddress, Headers, MessageID: hstring;
  Attachments: array of TOutlookMessageAttachment;
  function AsHTML: hstring; override;
  function Style: string; override;
end;

TOutlookMessageAttachment = record
  FileName, ShortFileName, CID, DisplayName: hstring;
  MIMEType: string;
  BinaryData: TBytes;
  RenderingPos: integer;
end;
```

12 CFF/TTF/WOFF

Library contains several classes designed to works with TTF,CFF, WOFF and PostScript fonts. These classes allows loading fonts from file or stream, converting from one format to another or to SVG and working with font glyphs.

12.1 Compact Font Format

```
TCFFFont = class
public
  Header: TCFFHeader;
  NameIndex, TopDictIndex, StringIndex, GlobalSubrIndex, CharStringsIndex, FontDictIndex: TCFFIndex;
  LocalSubrIndexes: array of TCFFIndex;
  TopDict: TCFFDict;
  PrivateDicts, FontDicts: array of TCFFDict;
  Name: string;
  Charset: array of record CharCode: word; Name: string end; //Glyph unicode code and name
  Encoding: array of word; //Glyph CID
  Ascent, Descent, LineGap, XHeight, CapHeight, UnderlinePosition, UnderlineThickness, ItalicAngle: inte
  FirstChar, LastChar: integer;
  Widths: array of word;
  FontData: TBytes;
  CharStringType: integer;
  DefaultWidthX, NominalWidthX: word;
  IsType2: boolean;
  CMAP, CharsetCMAP: THtCMap;
  IsCIDFont: boolean;
  FDRanges: array of packed record First: word; FDIndex: byte end;
  EncodingName: string;
  FontMatrix: THtMatrixData;
  procedure ReadIndex(ST: TStream; var Res: TCFFIndex);
  procedure LoadfromStream(ST: TStream);
  procedure LoadfromType1Stream(ST: TStream);
  function AverageCharWidth: integer;
  function CharStringLocalSubr(CharIndex: integer): TPSSubroutines; end;
```

Header

CFF header

NameIndex, TopDictIndex, StringIndex, GlobalSubrIndex, CharStringsIndex, LocalSubrIndex, FontDictIndex

CFF indexes

TopDict, PrivateDict, FontDict

Font dictionaries

Name

Font name

Charset

Unicode code and adobe glyph name for font glyphs

Encoding

Glyph CIDs

Ascent, Descent, LineGap, XHeight, CapHeight, UnderlinePosition, UnderlineThickness, ItalicAngle

Font properties in design units

Widths

Glyph widths in design units

CharStringType

Type of data in CharStrings dictionary.

DefaulfWidthX, NiminalWidthX

Default and nominal glyph width

IsType2

Font type - Type2 or Type1

CMAP

Font characted map (glyph encoding).

IsCIDFont

CID font flag

FontMatrix

Font glyph matrix

LoadfromStream

Load Type2 font from stream

LoadfromType1Stream

Load Type1 font from stream

AverageCharWidth

Average glyph width

CharStringLocalSubr

Returnd subroutine for glyph

12.2 OpenType font format

```
TOTFFont = class
public
  OffsetTable: TOTFOffsetTable;
  CMAPTable: TOTFCMAPTable;
  FontHeader: TOTFFontHeaderTable;
  HHeader: TOTFHHeaderTable;
  MaxProfile: TOTFMaxProfileTable;
  Names: TOTFNameTable;
  HMetrics: TOTFHMetricsTable;
  OS2: TOTFOS2Table;
  PostScript: TOTFPostScriptTable;
  LocaTable: TOTFLocaTable;
  GlyphTable: TOTFGlyphTable;
  GPOS: TOTFGPOSTable;
  GSUB: TOTFGSUBTable;
  Kern: TOTFKerningTable;
  CFFData: TBytes;
  TableRecs: array of TOTFTableRec;
  Name: string;
  class function NormalizeFont(const FontData: TBytes; AFont: THtBaseFont): TBytes;
  class function NormalizeHeader(const FontData: TBytes): TBytes;
  class function WofftoOTF(const FontData: TBytes; const FontName: string = ''): TBytes;
  constructor Create;
  destructor Destroy; override;
  procedure LoadfromStream(ST: TStream);
  procedure LoadfromFile(const AFileName: string);
  procedure DeleteTablebyType(Tag: cardinal);
  procedure Assign(ASource: TCFFFont);
  procedure SavetoStream(ST: TStream);
  procedure SavetoWOFFStream(ST: TStream);
  procedure LoadfromWOFFStream(ST: TStream);
  procedure LoadfromWOFFFile(const AFileName: string);
  procedure SavetoFile(const AFileName: string);
  procedure SavetoWOFFFile(const AFileName: string);
  function AsFontFace(const AFamilyName: string): string;
  procedure SavetoSVG(const FileName: string);
  function GetGlyphName(GID: integer): string;
  function GetGlyphs(const s: hstring): TWordDynArray;
  function CreateLayout(const Text: hstring; const FontSize: single; out Layout: THtLayout): boolean;
  procedure PrepareLayout(var Layout: THtLayout);
  procedure GetGlyphPlacement(var Layout: THtLayout);
  function MeasureString(const Text: hstring; const FontSize: single): single;
  function Ascent(const FontSize: single): single;
  function Descent(const FontSize: single): single;
  function Underline(const FontSize: single): single;
  property Tables[Index: integer]: TOTFTable read GetTable;
  property TableCount: integer read GetTableCount;
  property IsTTF: boolean read GetIsTTF write SetIsTTF;
  ///<summary> Returns font as set of SVG images</summary>
  property AsSVG: string read GetSVG; end;
```

OffsetTable, CMAPTable, FontHeader, HHeader, MaxProfile, Names, HMetrics, OS2, PostScript, LocaTable, GlyphTable, GPOS, GSUB, Kern

Font tables (may be nil).

CFFData

Font data when font is in CFF format

TableRecs

Font table records

Name

Font name

NormalizeFont

Add necessary tables and encodings to font.

NormalizeHeader

Calculate font header fields

WofftoOTF

Convert WOFF font to OTF font.

LoadfromStream

Load OTF from stream

LoadfromFile

Load OTF from file

DeleteTablebyType

Delete font table by table type

Assign

Load from CFF font.

SavetoStream

Save to OTF stream

SavetoWOFFStream

Save to WOFF stream

LoadfromWOFFStream

Load from WOFF stream

LoadfromWOFFFile

Load from WOFF file

SavetoFile

Save to OTF file

SavetoWOFFFile

Save to WOFF file

AsFontFace

Get base64 encoded string for embedding into HTML

SavetoSVG

Save all glyphs as SVG

GetGlyphName

Postscript glyph name

GetGlyphs

Get glyph IDs for string

CreateLayout

Create new layout object, set glyph IDs and set properties to default values

PrepareLayout

Process glyph substitutions, position adjustment and kerning (GPOS, GSUB and KERN tables)

GetGlyphPlacement

Set glyph widths

MeasureString

Measure string width

Ascent

Font ascent value

Descent

Font descent valie

Underline

Underline position (from top)

Table

Font tables by index

TableCount

Font table count

IsTTF

Font oulines type: TTF or CFF

AsSVG

Get font glyphs as SVG string.

12.3 Layout

```
THtLayout = record
public
  Font: TOTFFont;
  FontSize, LetterSpacing: single;
  Glyphs: array of THtLayoutGlyph;
  Flags: byte;
  StylisticSet: TOTFStylisticFeatures;
  FeatureSet: TOpentypeFeatures;
  FontStyle: THtFontStyles;
  function Width: single;
  function Ascent: single;
  function Descent: single;
  function AsSVG: string;
  procedure AsPath(P: THtPath);
  property RTL: boolean read GetRTL write SetRTL;
  property Kerning: boolean read GetKerning write SetKerning;
  property Substitution: boolean read GetSubstitution write SetSubstitution;
  property Positioning: boolean read GetPositioning write SetPositioning;
end;

THtLayoutGlyph = record
  Symbol code
  CID: cardinal;
  Glyph ID
  GID: word;
  Offset from default position
  DX, DY: Smallint;
  Width in font units
  Width: SmallInt;
end;
```

Glyphs

Layout glyphs. After calling PrepareLayout glyph number can change and will not correspond to initial text length.

Flags

Font flags. Use properties to get/set flags.

StylisticSet

Set of stylistic features (ss01..ss20) that should be processed during PrepareLayout

FeatureSet

Set of OpenType features that should be processed during PrepareLayout

FontStyle

Bold, Italic, Underline, Strikeout

Width

Total width of layout

Ascent

Ascent value

Descent

Descent value

AsSVG

Convert layout to SVG string

AsPath

Convert layout to SVG path

RTL

Layout contains right-to-left text

Kerning

Process kerning when prepare layout (Kern table)

Substitutions

Process glyph substitutions when prepare layout (ligatures, etc.GSUB table)

Positioning

Adjust glyphs positions when prepare layout (GPOS table)

12.4 Loading font from file or stream

To load font create font object

```
var F: TOTFFont;
F := TOTFFont.Create;
```

and call one of the following methods:

- LoadfromFile
- LoadfromStream
- LoadfromWOFFFile
- LoadfromWOFFStream

12.5 Saving font to file or stream

To save modified fonr use one of the following methods:

- SavetoFile
- SavetoStream
- SavetoWOFFFile
- SavetoWOFFStream

12.6 Measuring text width

Sample code:

```
Font := TOTFFont.Create;
try
  Font.LoadfromFile(FontFile); //or LoadfromStream
  W := Font.MeasureString(s, FontHeight);
finally
  Font.Free
end;
```

12.7 Converting text to SVG

Sample code

```
   Font := TOTFFont.Create;
   try
     Font.LoadfromFile('MyFont.ttf');
     Font.CreateLayout(Text, 40, L);
     Font.PrepareLayout(L);
     SVG := L.AsSVG('', 'fill="blue" stroke="green"');
  finally
    Font.Free
  end;
```

12.8 Custom HTML canvas example

Example of HTML canvas which use library font class for text measurement and rendering (only API used from DX is FillGeomatry).

```
type
  THtFontOTF = class(THtFont)
  private
    procedure FreePathCache;
  protected
    procedure GetMetrics;
  public
    Font: TOTFFont;
    PathCache: array of THtPath;
    constructor Create(const AOwner: THtFontCollection; const AFamily: string; const AHeight: single; ASt
  end;

  THtCanvasOTF = class(THtCanvasDX)
  public
    class function FontClass: THtFontClass; override;
    class function MeasureString(const s: string; const AFont: THtFont; const TextStyle: THtTextStyle;
      const Scale: single = 0; AFirstChar: integer = 1; ALength: integer = -1): single; override;
    procedure DrawString(const s: string; const R: TRectF; {$IFDEF AUTOREFCOUNT}[unsafe]{$ENDIF} const Fo
      const TextStyle: THtTextStyle;  AFirstChar: integer = 1; ALength: integer = -1); override;
    class function GetFlags: THtCanvasFlags; override;
  end;

constructor THtFontOTF.Create(const AOwner: THtFontCollection;
  const AFamily: string; const AHeight: single; AStyle: THtFontStyles);
begin
  inherited;
  GetMetrics;
end;

procedure THtFontOTF.GetMetrics;
var B: VCL.Graphics.TBitmap;
    T: TBytes;
    n: integer;
    ST: TBytesStream;
begin
  B := VCL.Graphics.TBitmap.Create;
  try
    B.Width := 1;
    B.Height := 1;
    FreeandNil(Font);
    B.Canvas.Font.Name := Family;
    if hfsItalic in Style then
      B.Canvas.Font.Style := B.Canvas.Font.Style + [fsItalic];
    if hfsBold in Style then
      B.Canvas.Font.Style := B.Canvas.Font.Style + [fsBold];
    B.Canvas.Font.Size := 10;
    n := GetFontData(B.Canvas.Handle, 0, 0, nil, 0);
    if n = -1 then
      RaiseLastOsError;
    SetLength(T, n);
    n := GetFontData(B.Canvas.Handle, 0, 0, pointer(T), n);
    ST := TBytesStream.Create(T);
    try
      Font := TOTFFont.Create;
      Font.LoadfromStream(ST);
    finally
      ST.Free
    end;
    Self.MinCharWidth := Font.MeasureString('i', Height);
    Self.Ascent := Font.Ascent(Height);
    Self.Descent := Font.Descent(Height);
    Self.LineHeight := Self.Ascent + Self.Descent;
    Self.SpaceWidth := Font.MeasureString(' ', Height);
    FreePathCache;
    SetLength(PathCache, Font.GlyphTable.Count);
    FillChar(pointer(PathCache)^, Length(PathCache) * sizeof(THtPath), 0);
  finally
    B.Free
  end;
end;

procedure THtFontOTF.FreePathCache;
var i: integer;
begin
```

```
  for i := 0 to High(PathCache) do
    if PathCache[i] <> nil then
      PathCache[i].Free;
  SetLength(PathCache, 0);
end;


procedure THtCanvasOTF.DrawString(const s: string; const R: TRectF;
  const Font: THtFont; const TextStyle: THtTextStyle; AFirstChar,
  ALength: integer);
var L: THtLayout;
    Scale: single;
    X, W: single;
    i, Offs: integer;
    B: THtBrush;
    F: THtFontOTF;
    M: THtMatrixData;
begin
  if ALength < 0 then
    ALength := Length(s);
  F := THtFontOTF(Font);
  F.Font.CreateLayout(copy(s, AFirstChar, ALength), Font.Height, L);
  F.Font.PrepareLayout(L);
  SaveState;
  M.Init(1, 0, 0, 1, R.Left, R.Top + round(F.Font.Ascent(F.Height) - F.Font.Ascent(F.Height)));
  FCurrentMatrix.Multiply(M);
  FContext.SetTransform(TD2D_MATRIX_3X2_F(FCurrentMatrix.Data));
  B := GetBrush(TextStyle.Color);
  Scale := Font.Height / F.Font.FontHeader.Header.UnitsPerEM;
  for i := 0 to High(L.Glyphs) do
  begin
    if F.PathCache[L.Glyphs[i].GID] = nil then
    begin
      F.PathCache[L.Glyphs[i].GID] := PathClass.Create;
      F.Font.GlyphTable.Glyphs[L.Glyphs[i].GID].AsPath(
       F.Font.GlyphTable, 0, 0, F.PathCache[L.Glyphs[i].GID], Scale);
    end;
  end;

  W := L.Width;
  if L.RTL then
  begin
    M.Init(1, 0, 0, 1, W, 0);
    FCurrentMatrix.Multiply(M);
    for i := 0 to High(L.Glyphs) do
    begin
      Offs := L.Glyphs[i].DX + L.Glyphs[i].Width;
      M.Init(1, 0, 0, 1, -Offs * Scale, 0);
      FCurrentMatrix.Multiply(M);
      FContext.SetTransform(TD2D_MATRIX_3X2_F(FCurrentMatrix.Data));
      FillPath(B, F.PathCache[L.Glyphs[i].GID]);
    end;
  end else
  begin
    for i := 0 to High(L.Glyphs) do
    begin
      Offs := L.Glyphs[i].DX;
      if i > 0 then
        Offs := Offs + L.Glyphs[i-1].Width;
      M.Init(1, 0, 0, 1, Offs * Scale, 0);
      FCurrentMatrix.Multiply(M);
      FContext.SetTransform(TD2D_MATRIX_3X2_F(FCurrentMatrix.Data));
      FillPath(B, F.PathCache[L.Glyphs[i].GID]);
      end;
  end;
  RestoreState;
  if hfsUnderline in F.Style then
    DrawLine(P, R.Left, R.Top + F.Font.Underline(F.Height), R.Right, R.Top + F.Font.Underline(F.Height));
end;

class function THtCanvasOTF.FontClass: THtFontClass;
begin
  Result := THtFontOTF;
end;
```

```
class function THtCanvasOTF.MeasureString(const s: string; const AFont: THtFont;
  const TextStyle: THtTextStyle; const Scale: single; AFirstChar,
  ALength: integer): single;
begin
  if ALength < 0 then
    ALength := Length(s);
  Result := THtFontOTF(AFont).Font.MeasureString(copy(s, AFirstChar, ALength), AFont.Height);
end;

class function THtCanvasOTF.GetFlags: THtCanvasFlags;
begin
  Result := [hcfScaleCanvasDrawing, hcfCacheTextWidth, hcfSupportsSimpleLayouts, hcfFullRepaint];
end;
```

13 Postscript

Library contains set of classes designed for executing Adobe Postscript programs. This class is used for converting PostScript fonts into TTF font oulines.

```
TPSProcessor = class
public
   Encoding, Subrs, OtherSubrs: TPSArray;
   SystemDict, GlobalDict, CharStrings, FontInfo, PrivateDict, FontDict: TPSDictionary;
  constructor Create(const AData: TBytes);
  destructor Destroy; override;
  procedure Process;
end;
```

Encoding, Subrs, OtherSubrs: TPSArray;

Subroutines and encodings

SystemDict, GlobalDict, CharStrings, FontInfo, PrivateDict, FontDict: TPSDictionary;

Postscript dictionaries

Process

Execute postscript program and parse result

14 EMF/WMF

THtEMFImageClass is designed for parsing and EMF and WMF files and converting into SVG.

```
THtEMFImage = class
public
  Header: TEMFHEADER;
  WMFHeader: TWMFHEADER;
  IsWMF: boolean;
  class function ConvertEMF(const AData: TBytes; const R: TRect; const AStyle: string = ''): string;
  constructor Create(AStream: TStream);
  destructor Destroy; override;
  function AsSVG(const R: TRect; const AStyle: string = ''): string;
end;
```

Header

EMF image header

WMFHeader

EMF image header

IsWMF

Image type: EMF or WMF

ConvertEMF

Convert EMF or WMF into SVG

AsSVG

Get image SVG

15 Search engine

Library contains set of classes designed for creating fully functional search engine..It is based on packed bitmap index and can be used for indexing large amount of documents (1 million and more). Engine core is optimized for low memory consumption and fast indexing and searching. Main features:

- Virtuall folders interfaces for processing compressed files, email databases and other storages
- Search query completion
- Morphology and normalization (removes diacritics, etc).
- Ordering by relevance or date
- Snippets
- Table of contents
- Date and type map for search query

Sample project is located in /Demos/SearchEngine/ folder (requires HTML Library bundle).

15.1 Virtual Folders

Virtual folder interface is designed for unifying access to different document storages. It allows to index and show documens stored inside other files. F.e. the following path can be used to show word document compressed to ZIP archive and attached to email:

c:\users\John\My Documents\Outlook Files\my@my.com\Inbox\2625\Test.zip/Sample.docx

IHtVirtualFolder

```
 IHtVirtualFolder = interface

   function Name: string;
Name without path

   function FullName: string;
Full name including path

   function Location: string;
Folder Location  (for displaying)

   function Modified: TDateTime;
Modification date (useful for containers)

   function FileSize: int64;
File size in bytes

   function IsFolder: boolean;
Is folder (container) or file

   function FindFirst: IHtVirtualFolder;
Get first child folder

   function FindNext: IHtVirtualFolder;
Get next child folder

   function Find(const AFileName: string): IHtVirtualFolder;
Get child folder by name

   function AsStream: TStream;
Get as stream

   function AsText: hstring;
Get plain text

   function Owner: IHtVirtualFolder;
Parent folder

   function AsObject: TObject;
Folder object
```

THtBaseVirtualFolder

THtBaseVirtualFolder is a base class for implementing most of virtual foder classes.

THtSystemFolder

Standard system folder

THtZIPFolder

ZIP archive

THtZipFile

File inside ZIP archive

THtRarFolder

RAR archive

THtRarFile

File inside RAR archive

TPSTFile

Outlook PST or OST container.

TPSTFolder

Folder inside Outlook container

TPSTMessage

Outlook message inside PST/OST

TPSTAttachment

Outlook message attachment

TPSTMessageBody

Outlook message body

THtBatFolder

The Bat folder file

THtBatMessage

Message inside The Bat container

THtBatMessageBody

The Bat message body

THtBatMessageAttachment

The bat message attachment.

15.2 Main class

```
 THtSearchEngine = class
   OnTranslate: THtGetVarEvent;
Translation callback (for UI).

   MorphDictionary: THtMorphDictionary;
Morphology dictionary

   Diacritics: THtDiacritics;
Diacritics processor

   ImageURL: string;
URL template for images inside documents

   ThumbItemTemplate: string;
Template of single thumbnail item

   ThumbTemplate: string;
Template of thumbnail page

   SearchResultTemplate: string;
Template of search results

   procedure BeginIndexing;
Called before indxing

   procedure EndIndexing;
Called after indexing

   function Search(const AQuery: hstring; FromIndex, ToIndex: integer; var Q: THtSearchQuery): THtStorage
Perform search for AQuery and return document IDS from FromIndex to ToIndex. Q should have set the follow
 AllWords
 StartDate
 EndDate
 FileTypes
 Order
   function SearchDocumentCount(const AQuery: hstring): integer;
Get document count for query

   function Translate(const s: string): string;
Translate text

   function GetAutocomplete(const AQuery: string): string;
Get autocompletion list in JSON format.

   function GetDateMap(const Query: hstring; AllWords: boolean; const Types: string): hstring;
Get map of date/document count for Query

   function GetTypeMap(const Query: hstring; AllWords: boolean; const Types: string): hstring;
Get map of type/document count for Query

   function GetDocumentImage(DocID, RoleID: integer; const ImageID: string): TBytes; virtual;
Get image from document

   function GetDocumentThumbnail(ADocID, AWidth, AHeight: integer): TBytes; virtual;
Get document thumbnail

   function GetThumbnails(const Query, FileTypes, Sort: string; AllWords: boolean; RoleID: integer): stri
Get thumbnail page

   function GetDocument(DocID, RoleID: integer; const SearchQuery: string; PagedLayout: boolean = false):
Get document HTML

   function GetSearchResult(const Query, FileTypes, SortBy: string; AllWords: boolean;
      RoleID: integer; AddQuote: boolean = true; const DateRange: string = ''): hstring;
Perform search for AQuery and return documents using SearchTemplate.

   function CreateTOC(const s: string): string;
Create table of contents from HTML

   function GetTOC(ADocID, ARoleID: integer): hstring;
Get Table of Contents from document
```

```
   function GetSnippets(ADocID, ARoleID: integer; const AQuery: hstring): hstring;
Get document snippet

   function GetPDF(ADocID, ARoleID: integer): TBytes;
Convert document to PDF

   procedure ClearIndex;
Clear text index.

   property FileTypes: THtFileTypes read FFileTypes;
List of indexed types

   property DocumentDatabase: THtDocumentDatabase read FDocumentDatabase;
Document database.
   property FolderFactory: THtFolderClassFactory read FFolderFactory;
Folder factory for registering folder clases.
```

15.3 Document Database

In order to store indexed documents, Search Engine should have DocumentDatabase instance. It have the following methods;

```
 THtDocumentDatabase = class
 public
   procedure GetFolders(out AFolders: THtDocumentFolders);
Get list of ndexed folders (each document is linked to top level folder).

   function GetDocumentbyFileName(const AFileName: string; out DocID: integer; out AUpdated: TDateTime):
Search for document by its full name.

   function AddDocument(const AFileName, ALocation, AFileType, ADocumentName, ATitle: string;
     FolderID: integer; ADocument: THtStorageDoc): integer;
Add document to database and return ID

   procedure DeleteDocument(DocID: integer);
Delete document.

   procedure UpdateDocument(ADocument: THtStorageDoc); virtual; abstract;
Update document attributes.

   procedure SetDocumentError(DocID: integer; const AMessage: string); virtual; abstract;
Set Error filed for document.

   function GetDocumentbyId(DocID: integer; var FullName, FileType, FileName, Title: string; var FolderId
Get document attributes by ID

   function CheckAccess(RoleID, FolderID: integer): boolean; virtual;
Check if this role has access to documents in folder FolderID.
```

16 Extracting plain text

Some document classes has special optmized mode for extracting plain text without reading whole document. Code sample:

```
var OD: THtOfficeDocument;

 OD := SomeOfficeDocumentClass.Create
 if OD.SupportsTextMode  then
 begin
    OD.TextMode := true;
    OD.LoadFromFile(AFile);
    text := OD.AsText;
  end
```

For other classes use

OD.AsText

Also THtDocumentConverter class contains general method for extracting text content from any document without creating document instance:

```
  class function FiletoText(const AFileName: string; AOnText: THtTextCallback = nil): hstring;
  class function StreamtoText(const AStream: TStream; AOnText: THtTextCallback = nil): hstring;
```

17 Password protected documents

Library supports reading of password protected documents of the following types: **PDF, DOCX, XLSX, PPTX.**
Protection is supported only on Windows platform in Delphi versions **XE2** and newer.
There are two ways of passing password:
1.Set Document.**Password** property before reading document
2.set Document.**OnPasswordRequest** property (event) which be called when loading protected file.

THtPasswordRequestEvent = **function**(Sender: TObject; **var** Password: hstring): boolean **of object**;

Method should set **Password** parameter and return true.

Also this event can be passed as parameter to **IHtDocumentConverter** methods lile **ConvertFile** or **ConvertStream**.

Note: when reading protected documents there will be (caught) exception, this is normal behavior.

18 Image handling

By default images are embedded into document, This simplifies conversion process but can lead to significant memory consumption on large documents.
Other option is to embed only link to image and provide image via callback:

1. Set EmbedImages to false
2. Set ImageURL to some value, f.e './docimages?doc_id=123&image=';
3. Use Document.GetImageData to return image data in callback (HtPanel.OnGetImage or web server)

Index

- V -

```
var OD: THtOfficeDocument;
    begin     64
    end     64
    if OD.SupportsTextMode  then     64
    OD := SomeOfficeDocumentClass.Create     64
    OD.LoadFromFile(AFile);     64
    OD.TextMode := true;     64
    text := OD.AsText;     64
```