



Getting started with

# **ScroogeXHTML for Delphi**

*Version 6.11*

### **Trademarks**

Habari is a trademark or registered trademark of Michael Justin in Germany and/or other countries. Android is a trademark of Google Inc. Use of this trademark is subject to Google Permissions. The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License. Embarcadero, the Embarcadero Technologies logos and all other Embarcadero Technologies product or service names are trademarks, servicemarks, and/or registered trademarks of Embarcadero Technologies, Inc. and are protected by the laws of the United States and other countries. IBM and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries, or both. HornetQ, WildFly, JBoss and the JBoss logo are registered trademarks or trademarks of Red Hat, Inc. Mac OS is a trademark of Apple Inc., registered in the U.S. and other countries. Oracle, WebLogic and Java are registered trademarks of Oracle and/or its affiliates. Pivotal, RabbitMQ and the RabbitMQ logo are trademarks and/or registered trademarks of GoPivotal, Inc. in the United States and/or other countries. Other brands and their products are trademarks of their respective holders.

## Contents

<b>Introduction.....</b>	<b>5</b>
<b>About ScroogeXHTML.....</b>	<b>5</b>
Features.....	5
Limitations.....	5
API Documentation.....	5
ScroogeXHTML for the Java™ Platform.....	5
<b>Installation.....</b>	<b>6</b>
<b>Requirements.....</b>	<b>6</b>
<b>Component installation in Delphi.....</b>	<b>6</b>
Prepared Packages.....	6
<b>Compatibility.....</b>	<b>7</b>
<b>ConvertUsingPrettyIndents Property.....</b>	<b>7</b>
<b>DocumentType Property.....</b>	<b>7</b>
<b>Conversion Methods.....</b>	<b>8</b>
<b>RTF to XHTML Conversion Methods.....</b>	<b>8</b>
Simple conversion example using the Convert function.....	8
ConvertRTFFile.....	10
ConvertRTF.....	10
RichEditToHTML.....	10
<b>Configuration Options.....</b>	<b>11</b>
<b>Document Related Properties.....</b>	<b>11</b>
Document Type.....	11
<b>Font Related Properties.....</b>	<b>12</b>
Font Name Replacement.....	13
Font Size Scale.....	13
<b>Special Character related Options.....</b>	<b>13</b>
Conversion of Space Characters.....	13
Tab characters.....	14
<b>Paragraph related Options.....</b>	<b>14</b>
Empty Paragraphs.....	14
Indentation.....	14
<b>Advanced Configuration.....</b>	<b>15</b>
<b>Embedding HTML Output in existing Documents.....</b>	<b>15</b>
<b>Hyper Links.....</b>	<b>16</b>
How Hyper Links are detected.....	16
Check List.....	17
Automatic Hyper Link Handling.....	17
E-Mail Links.....	17
HTTP Links.....	17

HyperlinkList.....	17
OnHyperlink Event Handler.....	18
<b>Default Font Styles.....</b>	<b>18</b>
OptionsOptimize Property Group.....	18
<b>ElementStyles and ElementClasses.....</b>	<b>20</b>
List of supported Elements.....	20
ElementStyles.....	20
ElementClasses.....	21
<b>Picture Support.....</b>	<b>22</b>
<b>Introduction.....</b>	<b>22</b>
<b>Requirements.....</b>	<b>22</b>
<b>IPictureAdapter Interface.....</b>	<b>23</b>
Purpose.....	23
Usage Example.....	23
<b>Picture Data Conversion.....</b>	<b>23</b>
WMF Picture Converter.....	24
<b>Web Demo.....</b>	<b>25</b>
<b>International Support.....</b>	<b>28</b>
<b>Unicode and Code Pages.....</b>	<b>28</b>
Requirements on the Client Side.....	28
Requirements on the Producer Side.....	28
<b>Left-to-right and right-to-left text direction.....</b>	<b>28</b>
<b>Language Attributes.....</b>	<b>28</b>
<b>Logging.....</b>	<b>29</b>
<b>Configuration of Log Messages.....</b>	<b>29</b>
Usage of the OnLog event handler.....	29
Setting the log level.....	29
Logging over other logging frameworks.....	30
<b>Frequently Asked Questions.....</b>	<b>31</b>
<b>Conversion.....</b>	<b>31</b>
How can I remove the space between lines?.....	31
How can I remove the indentation?.....	32
<b>Index.....</b>	<b>33</b>

# Introduction

---

## About ScroogeXHTML

### Features

**ScroogeXHTML for Delphi** converts text attributes including background and highlight colors, paragraph attributes including alignment (left, right, centered, justified) and paragraph indent (left, right, first line) and simple numbered or unnumbered lists. Unicode conversion allows international documents, including simplified and traditional Chinese, Korean and Japanese. CSS and default font settings allow to create optimized documents. Supported document types:

- XHTML 1.0 Strict and Transitional
- XHTML Basic 1.0
- XHTML Mobile Profile 1.0 (aka WAP 2.0)
- HTML 4.01 Strict and Transitional
- HTML5

### Limitations

The RTF specification contains very many elements and features, ScroogeXHTML does only convert a limited subset.

### API Documentation

The API documentation can be found in the installation folder, a link to the current on-line version can be found at [https://www.habarisoft.com/scroogexhtml\\_delphi.html](https://www.habarisoft.com/scroogexhtml_delphi.html)

### ScroogeXHTML for the Java™ Platform

ScroogeXHTML is also available for the Java™ platform.

## Installation

---

### Requirements

ScroogeXHTML supports the following development environments:

- Delphi 2009 or newer
- Free Pascal 2.6.4

---

### Component installation in Delphi

#### **Time saving tip**

Installing ScroogeXHTML is not required. The component works fine if you simply add the source path to your project. This will save you the work of installing it for every new version of the component and new installations of Delphi

To install ScroogeXHTML in the Delphi component palette, follow these steps:

#### **1. create a new Delphi Package Project**

- in the IDE menu, select "File | New | Package - Delphi"

#### **2. add the file ScroogeXHTML\_reg.pas**

- in the IDE menu, select "Project | Add to Project ..."
- choose the file <Scrooge Install Folder>\source\ScroogeXHTML\_reg.pas
- click on Open

#### **3. install the package**

- in the project manager view, right-click on the package node
- choose "Install"

The component will appear in a new palette page with the title 'Habarisoft'.

### Prepared Packages

The source code distribution of ScroogeXHTML contains prepared packages in the *packages* folder for your convenience.

# Compatibility

---

## ConvertUsingPrettyIndents Property

New since release 4.4 is a new property which can be set to False to enable a workaround for a rendering problem in Internet Explorer.

```
SX.ConvertUsingPrettyIndents := False;
```

Note: The default is True so that the output is backward compatible with previous component versions.

If the property is set to False, the generated output document will not use indentation of closing paragraph tag (`</p>`). Instead, the `</p>` tag will immediately follow the preceding `</span>` tag.

---

## DocumentType Property

If the conversion result should have 'almost WYSIWYG'-style output quality, it is recommended to use the Transitional document types which will cause better rendering results in many web browsers.

- 'HTML 4.01 Transitional'
- 'XHTML 1.0 Transitional'

Also, the new HTML5 standard is recommended if compatibility with older browsers is not required.

## Conversion Methods

---

### RTF to XHTML Conversion Methods

One low-level function (declared in the **TSxMain** class) and three high-level conversion methods (declared in the **TCustomScrooge** class) are provided by the component.

<b>Convert</b>	This function converts a string containing RTF code and returns a HTML output string (UTF8String). This function is useful for in-memory conversions, for example when the RTF code is read from a database table and the result HTML code is sent to a web browser client
<b>ConvertRTFFile</b>	This procedure converts an existing RTF file and saves the result HTML code in an output file
<b>ConvertRTF</b>	This function converts an existing RTF file and returns a HTML string (UTF8String)
<b>RichEditToHTML</b>	this function converts the RTF code in a TRichEdit component to HTML (UTF8String)

### Simple conversion example using the Convert function

This example shows how to use the Convert method to write the HTML code to standard output. The output will be generated using the default configuration of the component.

```
program Example1;
{$APPTYPE CONSOLE}
uses
  ScroogeXHTML;
var
  SX: TBTScroogeXHTML;
begin
  SX := TBTScroogeXHTML.Create;
  try
    WriteLn(SX.Convert('{\rtf1 {\b bold \i Bold Italic \i0 Bold
again}}'));
  finally
    SX.Free;
  end;
  ReadLn;
end.
```

The example also shows how to create a component instance at run time. In some applications it is necessary to create instances of components at run time - for example,



multi threaded applications such as web servers require one component instance per thread.

The generated output:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
      Untitled document
    </title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <meta name="generator" content="TBTScoogeXHTML 6.7" />
  </head>
  <body>
    <p>
      <span style=
        "font-weight:bold;">bold </span><span style=
          "font-weight:bold;font-style:italic;">Bold Italic </span><span
style=
  "font-weight:bold;">Bold again</span>
    </p>
  </body>
</html>
```

This example shows that the default component settings will generate a document with the XHTML 1.0 document type and a head section with default values for title and meta tags.

Let's take a closer look at the result code:

- the first line (`<?xml version="1.0"?>`) declares that the document is a XML 1.0 file. This line is optional and will be included by default for XHTML based document types.
- the second line is the document type declaration which will be used by HTML and XHTML parsers and validators to verify that the document's syntax is correct. The document type can be set with the `DocumentType` property.
- the `<html>` element includes an attribute which declares the XHTML name space, this attribute will included automatically in XHTML based documents.
- the `<head>` section contains a document title element, the `DocumentTitle` property can be used to set its content.
- the content-type `<meta>` tag will be generated automatically and tells the browser that the document uses the UTF-8 encoding.
- the body contains a `<p>` (paragraph) element which uses `<span>` elements to apply styles to the text parts with bold and bold/italic format.

Important notes:

- It is possible to use the component to generate only the body part, without all HTML elements which declare the head and the start of the body section. See section `AddOuterHTML` for more details.

## **ConvertRTFFile**

This method converts an existing RTF file and saves the result in an output file.

## **ConvertRTF**

This method converts an existing RTF file and saves the result in a result string.

## **RichEditToHTML**

This method converts the RichEdit RTF code to HTML/XHTML.

# Configuration Options

---

## Document Related Properties

### Document Type

The `DocumentType` property selects the output document type. There are two flavors of output documents: HTML based and XHTML based.

#### HTML based

- HTML 4.01 Transitional
- HTML 4.01 Strict
- HTML5

#### XHTML based

- XHTML 1.0 Transitional
- XHTML 1.0 Strict
- XHTML Basic 1.0
- XHTML Mobile 1.0
- XHTML 1.1

Depending on the document type, some HTML/XHTML elements are not supported. For example, transitional HTML and XHTML will allow more elements and attributes than their Strict counterparts.

#### Example code

In this example, the document type will be HTML 4.01 Transitional:

```
...
SX := TBTScroogeXHTML.Create;
try
  SX.DocumentType := dtHTML_401_Transitional;
  WriteLn(SX.Convert(EXAMPLE_RTF));
finally
  SX.Free;
end;
...
```

## Font Related Properties

### FontConversionOptions

The conversion of font size, font name and other character properties can be controlled with the FontConversionOptions property. This property is a set of switches:

<b>opFontSize</b>	enables conversion of font sizes
<b>opFontName</b>	enables conversion of font names. See also TCustomScrooge.ReplaceFonts
<b>opFontStyle</b>	enables conversion of font styles (bold, italic, underlined, strikeout)
<b>opFontColor</b>	enables conversion of font colors
<b>opFontBGColor</b>	enables conversion of background font colors
<b>opFontHLCOLOR</b>	enables conversion of highlight font colors

By default all options are enabled except `opFontHLCOLOR`. If you want to activate only a subset of these options, your code has to assign a list of these options to the FontConversionOptions property (e. g. `[opFontSize, opFontName]`).

### Example code

In this example, the FontConversionOptions are set to an empty list

```
program Example2;
{$APPTYPE CONSOLE}
uses
  ScroogeXHTML, SxTypes;
const
  EXAMPLE_RTF = '{\rtf1 {\b bold \i Bold Italic \i0 Bold again}}';
var
  SX: TBTScroogeXHTML;
begin
  SX := TBTScroogeXHTML.Create;
  try
    SX.OptionsHead.AddOuterHTML := False;
    SX.FontConversionOptions := [];
    WriteLn(SX.Convert(EXAMPLE_RTF));
  finally
    SX.Free;
  end;
  ReadLn;
end.
```

The resulting output does not include the font style tag

```
<p>
  bold Bold Italic Bold again
</p>
```

## Font Name Replacement

If RTF documents use fonts which are not available in the HTML browser, the look of the converted document is unpredictable. In some cases the document, the font names can be manually changed to more common ones. But there might be documents which should be left unmodified. As a workaround, the component offers a font name replacement option which uses a list of replacement rules.

The **ReplaceFonts** property contains these default replacement rules:

<b>Arial</b>	all font names starting with "Arial" will be replaced by "Arial,Helvetica,sans-serif"
<b>Courier</b>	all font names starting with "Courier" will be replaced by "Courier,monospace"
<b>Symbol</b>	all font names starting with "Symbol" will be replaced by "Symbol"
<b>Times</b>	all font names starting with "Times" will be replaced by "Times,serif"

The replacement rules are key-value pairs. The following example shows how a new set of replacement rules can be defined.

```
...
SX.ReplaceFonts.Clear;
SX.ReplaceFonts.Add('Arial=sans-serif');
SX.ReplaceFonts.Add('Courier=monospace');
SX.ReplaceFonts.Add('Times=serif');
...
```

## Font Size Scale

The **FontSizeScale** property sets the font size scale. The following units are supported:

<b>point (pt)</b>	Font sizes are expressed in point (pt). This is the default property value. Point is an absolute size scale, all others are relative scales.
<b>em</b>	Relative font sizes, expressed in em units.
<b>ex</b>	Relative font sizes, expressed in ex units.
<b>percent</b>	Relative font sizes, expressed in percent values.

---

## Special Character related Options

### Conversion of Space Characters

In HTML browsers, there is no visual difference between a single space character and a sequence of two or more space characters.

Some RTF documents however make use of sequences of space characters for special visual effects, for example in combination with a fixed-space font like Courier.

These documents would look corrupt when they are converted to HTML. As a workaround, a Boolean property, **ConvertSpaces**, can be used to replace sequences of space characters by “nonbreakable spaces” (&nbsp;).

The ConvertSpaces property is set to False by default.

## Tab characters

HTML does not support the “Tab” character. However, this character may appear in RTF documents.

As a workaround, the component replaces tab characters by a sequence of non breakable spaces. The **TabString** property can be used to modify the replacement string.

---

## Paragraph related Options

### Empty Paragraphs

Set the **ConvertEmptyParagraph** property to True to replace empty paragraphs, where the opening <p> tag is followed by the closing </p> tag by a line break tag (<br />).

### Indentation

Set the **ConvertIndent** property to True if you want to activate support for left and right paragraph indents.

The ConvertIndent property is set to False by default.

**Note** the right indent in the output document is relative to the browser window - if you change the browser window size, the text area will adjust its size

## Advanced Configuration

---

### Embedding HTML Output in existing Documents

In many cases, the result of the RTF to HTML conversion shall not be a stand-alone document but should be included in existing HTML code.

For example, the component may be used to convert RTF code which is stored in a database table to build one HTML document with all the converted table records. This will however not be easy if the resulting HTML contains the `<head>` and `<body>` elements – a HTML document can only contain one `<head>` and one `<body>` section.

A master document, where we want to include generated HTML code somewhere in the body section, could look like this:

```
<html>
  <head>
    <title>
      Business report for 2013
    </title>
  </head>
  <body>
    <h1>
      Summary
    </h1>

    ... HTML generated by ScroogeXHTML should go here ...

  </body>
</html>
```

In this case it would be much easier if the HTML code generated by ScroogeXHTML only contained the part in the `<body>` section, so that we could insert it with in the master document using a simple string concatenation.

### AddOuterHTML Property

The OptionsHead property group controls the generation of HTML head and body elements, including the document title, META tags, and CSS stylesheet settings.

The AddOuterHTML property controls if the output will be a standalone HTML document or just a HML fragment which can be included in an existing HTML document.

- Set this property to True (which is the default value) to create output documents with surrounding tags for the HTML header and body
- Set this property to False to create output documents without surrounding tags for the HTML header and body

### Example code

This example sets the **AddOuterHTML** property to false.

```
program Example3;
{$APPTYPE CONSOLE}
uses
  ScroogeXHTML;
const
  EXAMPLE_RTF = '{\rtf1 {\b bold \i Bold Italic \i0 Bold again}}';
var
  SX: TBTScroogeXHTML;
begin
  SX := TBTScroogeXHTML.Create;
  try
    SX.OptionsHead.AddOuterHTML := False;
    WriteLn(SX.Convert('{\rtf1 {\b bold \i Bold Italic \i0 Bold
again}}'));
  finally
    SX.Free;
  end;
  ReadLn;
end.
```

The generated output does not include the `<html>` and `<head>` tags and looks like this:

```
<p>
  <span style=
    "font-weight:bold;">bold </span><span style=
    "font-weight:bold;font-style:italic;">Bold Italic </span><span
style=
    "font-weight:bold;">Bold again</span>
</p>
```

This code now could be used to embed it in an existing HTML document.

---

## Hyper Links

Hyper link support can be used to build links to other web pages. The component will include the necessary HTML element `<a href="...">(visible text)</a>` in the document.

### How Hyper Links are detected

If a piece of text uses **blue and underlined** formatting, the component will process it as a hyper link. The option `ConvertHyperlinks` enables or disables hyper link processing.



## Check List

To process hyper links, please verify that

- the property `ConvertHyperlinks` is set to `True`
- the property `FontConversionOptions` contains `opFontStyle` and `opFontColor`
- the hyper links in the document are formatted blue and underlined

### Example code

This example activates hyper link processing and makes sure that font styles and font colors will be detected.

```
...
SX.ConvertHyperlinks := True;
SX.FontConversionOptions := [opFontStyle, opFontColor];
...
```

## Automatic Hyper Link Handling

If hyper link conversion is enabled, and no `OnHyperlink` event handler has been defined, the component will process the blue and underlined text as a hyper link using its built-in default implementation for hyper link processing.

The component will not check if the blue and underlined text is a valid URL (Internet Address).

## E-Mail Links

If the property `HyperlinkOptions` contains `hoReplaceEMailLinks`, and the blue/underlined text contains the @ sign, the component will insert an email link in the output document.

## HTTP Links

If the property `HyperlinkOptions` contains `hoRequireHTTP`, and the blue/underlined text does not start with 'http:' or 'https:', the link will be ignored.

## HyperlinkList

The `HyperlinkList` property allows the converter to replace target addresses automatically by a hyper link. The display text will be unchanged.

### Example code

This example replaces the word `Foo` by a hyper link:

```
...
```

```
memHyperlinkList.text := 'Foo=http://www.example.com';  
..
```

## OnHyperlink Event Handler

The developer can write code for the OnHyperlink event to get total control over the hyperlink processing.

### Example code

The following example shows how to assign and use an event handler:

```
procedure TConverterTest.OnHyperlink(Sender: TObject; const TextElement:  
ISimpleDomTextNode);  
begin  
    TLinkURIBuilder.Build(TextElement);  
end;  
...  
procedure TConverterTest.Test;  
var  
    SX: TBTScroogeXHTML;  
begin  
    SX := TBTScroogeXHTML.Create;  
    try  
        SX.ConvertHyperlinks := True;  
        SX.FontConversionOptions := [opFontStyle, opFontColor];  
        SX.OnHyperlink := OnHyperlink1;  
  
        ...  
    finally  
        SX.Free;  
    end;  
end;
```

---

## Default Font Styles

### OptionsOptimize Property Group

The properties

- IncludeDefaultFontStyle
- DefaultFontName
- DefaultFontColor
- DefaultFontSize

are useful to create smaller HTML documents. The optimization method uses a CSS definition in the HEAD section, which defines the default font settings in the document. The component will only create font properties if a text block has a different style.

### Example

A simple RTF document (for example, created with WordPad) uses 'Times,serif' 10 pt as the main font. With the default settings, the component will create the full CSS style definition for every text block:

```
<p>
  <span style="
    font-family:Times,serif;color:#000000;font-size:10pt;">Hello
World</span>
</p>
```

To define and use the default CSS definition, use the following code:

```
...
SX.OptionsOptimize.DefaultFontName  := 'Times,serif';
SX.OptionsOptimize.DefaultFontColor := '#000000';
SX.OptionsOptimize.DefaultFontSize  := 10;
SX.OptionsOptimize.IncludeDefaultFontStyle := True;
SX.OptionsHead.AddOuterHTML := True;
...
```

The HEAD section of the generated document will now look like this:

```
<head>
  <title>
    Untitled document
  </title>
  <meta http-equiv="content-type" content="text/html; charset=iso-
8859-1">
  <style type="text/css">
    <!--
      BODY (font-family:Times,serif;font-size:10pt;color:#000000; )
    -->
  </style>
</head>
```

And the HTML code for the paragraph will now be reduced to:

```
<p>
  Hello World
</p>
```

---

## ElementStyles and ElementClasses

Two properties can be used to modify the generated HTML output on the element (or 'tag') level, they will add a style or class attribute to all elements with a given type.

Both properties are simple TStrings lists, and usually will be modified using the Values method:

### Example code

```
...  
SX.ElementStyles.Values['p'] := 'foo';  
SX.ElementStyles.Values['br'] := 'bar';  
...
```

## List of supported Elements

The following elements are supported:

- p (paragraph)
- br (line break)
- ol (ordered list)
- ul (bullet list)

## ElementStyles

The property ElementStyles is useful to define the appearance of paragraphs and other elements when there is no CSS definition in the document HEAD section, or when the CSS definition can not be changed (for example if the converted HTML fragment is only embedded in a HTML page).

### Example code

Define a style for the paragraph element to set the top and bottom margin to 0 pixel.

```
...  
SX.ElementStyles.Values['p'] := 'margin-bottom:0px;margin-top:0px;';  
...
```

The result HTML code for all paragraphs will now be:

```
<p style="margin-bottom:0px;margin-top:0px;">
```

## ElementClasses

The property `ElementClasses` assigns the class parameter for paragraphs and other elements. This allows to define more than one CSS definition for the paragraph element in the HTML document.

### Example code

Set the class parameter for all paragraph elements to 'id1'

```
...  
  
SX.ElementClasses.Values['p'] := 'id1';  
  
...
```

The result HTML code for all paragraphs will now be:

```
<p class="id1">
```

## Picture Support

---

### Introduction

ScroogeXHTML 5.1 introduces a new picture support feature, which is more flexible than the previous implementation. With this new implementation, it is possible to

- collect raw picture for embedded pictures
- access raw picture data as a TStream

In addition, the component also includes an example converter for embedded WMF pictures, which converts the picture data to PNG.

---

### Requirements

To extract data for embedded pictures, two properties of the component need to be set:

<b>ConvertPictures</b>	This Boolean property is false by default. The component will only convert pictures if it is set to true.
<b>PictureAdapter</b>	This property of type IPictureAdapter is not assigned by default.

Example:

```
uses
  ScroogeXHTML, SxInterfaces, SxMemoryPictureAdapter, (...)
var
  SX: TBTScroogeXHTML;
  PictureAdapter: IPictureAdapter;
begin
  SX := TBTScroogeXHTML.Create;
  try
    SX.ConvertPictures := True;
    PictureAdapter := TMemoryPictureAdapter.Create;
    SX.PictureAdapter := PictureAdapter;
    HTML := SX.ConvertRTF(RtfFile);
  finally
    SX.Free;
  end;
end;
```

---

## IPictureAdapter Interface

### Purpose

The `IPictureAdapter` interface defines methods for extraction of picture data. Most methods in this interface are only required for the internal data processing. The only exception is the `getPictures` function.

**getPictures** Returns a `TStrings` instance with all embedded pictures

### Usage Example

This example uses the `PictureAdapter.getPictures` function to retrieve the picture data and save it to disk.

```
var
  I: Integer;
  Pics: TStrings;
  E: TEmbeddedPicture;
begin
  // configure the component, convert and save the HTML
  ...

  // get the picture list
  Pics := SX.PictureAdapter.getPictures;

  // save pictures
  for I := 0 to Pics.Count - 1 do
  begin
    E := Pics.Objects[I] as TEmbeddedPicture;
    with TMemoryStream.Create do
      try
        LoadFromStream(E.Stream);
        SaveToFile(Pics[I]);
      finally
        Free;
      end;
    end;
  end;

  ...
end;
```

---

## Picture Data Conversion

By default, picture data will only be extracted but not converted to a web image format like JPEG, GIF or PNG.

For picture format conversion, the `IPictureAdapter` interface provides an optional method, `SetConverter`, which takes a parameter of type `IPictureConverter`.

## WMF Picture Converter

A basic implementation of a picture converter which can convert embedded WMF pictures to PNG is included.

```
...  
  
SX := TBTScroogeXHTML.Create;  
  
// configure the component  
...  
  
// assign picture adapter and WMF converter  
if ConvertPictures then  
begin  
    SX.PictureAdapter := TMemoryPictureAdapter.Create;  
    SX.PictureAdapter.SetConverter(TPictureConverterWMF.Create);  
end;  
  
...  
  
// run the conversion  
...
```



## Web Demo

The web demo application runs as a local web server (on port 80) and launches the web browser to display a HTML form where RTF code can be pasted and submitted to a ScroogeXHTML converter. The result HTML then will be displayed in the browser.

```

program SxWebDemo;

{$APPTYPE CONSOLE}

uses
  ScroogeXHTML, SxTypes,
  IdHTTPServer, IdCustomHTTPServer, IdContext,
  SysUtils, Classes, ShellAPI, Windows;

type
  TScroogeWebDemo = class(TObject)
  private
    Server: TIdHTTPServer;
    SX: TBTScroogeXHTML;
    procedure OnCommandGet(AContext: TIdContext;
      ARequestInfo: TIdHTTPRequestInfo; AResponseInfo:
      TIdHTTPResponseInfo);
  public
    constructor Create;
    procedure Run;
  end;

const
  IndexHtml = '<!DOCTYPE html>\n'
    + '<html>\n'
    + '  <head>\n'
    + '    <title>ScroogeXHTML Web Demo</title>\n'
    + '  </head>\n'
    + '  <body>\n'
    + '    <h1>ScroogeXHTML Web Demo</h1>\n'
    + '    <form method="post" action="/scroogedemo" accept-charset="US-ASCII">\n'
    + '      <fieldset>\n'
    + '        <legend>RTF Input</legend>\n'
    + '        <textarea name="rtf" rows="6" cols="60" style="margin: 0pt; width: 100%;">\n'
    + '      {\rtf1\n'
    + '      Hello! \par\n'
    + '      {\i This} is formatted {\b text}. \par\n'
    + '      Convert me.\n'
    + '    }\n'
    + '      </textarea><br />\n'
    + '    </fieldset>\n'

```

## 26 ScroogeXHTML for Delphi 6.11

```
+ '      <p><input type="submit" /></p>\n'
+ '    </form>\n'
+ '  </body>\n'
+ '</html>';

var
  Demo: TScroogeWebDemo;

  { TScroogeWebDemo }

constructor TScroogeWebDemo.Create;
begin
  SX := TBTScroogeXHTML.Create;
  Server := TIdHTTPServer.Create;
  Server.OnCommandGet := OnCommandGet;

  WriteLn('Server ready - using ScroogeXHTML ' + SX.Version);
end;

procedure TScroogeWebDemo.OnCommandGet(AContext: TIdContext;
  ARequestInfo: TIdHTTPRequestInfo; AResponseInfo: TIdHTTPResponseInfo);
var
  Rtf: RawByteString;
  Response: TStream;
begin
  if ARequestInfo.Command = 'GET' then
  begin
    if ARequestInfo.Document = '/' then
    begin
      Response := TStringStream.Create(StringReplace(IndexHtml, '\n', #13,
        [rfReplaceAll]));
      AResponseInfo.ContentStream := Response;
    end;
  end
  else if ARequestInfo.Command = 'POST' then
  begin
    if ARequestInfo.Document = '/scroogedemo' then
    begin
      Rtf := RawByteString(ARequestInfo.Params.Values['rtf']);
      Response := TStringStream.Create(
        {$IFDEF FPC}
          AnsiString(ToBytes(SX.Convert(Rtf)))
        {$ELSE}
          ToBytes(SX.Convert(Rtf))
        {$ENDIF}
      );
      AResponseInfo.ContentStream := Response;
    end;
  end;
end;

procedure TScroogeWebDemo.Run;
begin
  Server.Active := True;
end;

begin
  Demo := TScroogeWebDemo.Create;
  Demo.Run;
```

```
// launch browser
ShellExecute(0, 'open', 'http://localhost/', '', '', SW_SHOWNORMAL);

WriteLn('Hit any key to stop the server');
ReadLn;
end.
```

## International Support

---

### Unicode and Code Pages

Unicode conversion allows international documents, including simplified and traditional Chinese, Korean and Japanese.

ScroogeXHTML uses Unicode to create the output HTML document. This allows to include (and mix) many different languages in one HTML page.

### Requirements on the Client Side

On the client side, this solution works only if the HTML client (browser) supports Unicode and the necessary Unicode-enabled fonts are available on the system.

### Requirements on the Producer Side

Some RTF documents already contain Unicode encoded characters, which need no further processing in the converter component. They use the numeric Unicode values which can be translated immediately to HTML.

Many RTF documents however use national 'code pages' to encode special characters. If the component runs on a computer system which has no support for these code pages installed, the conversion of these RTF documents will not work.

Disclaimer: The component uses mappings between code pages and character sets which might be incomplete or wrong. There is no guarantee that the code page conversion always works as expected.

---

### Left-to-right and right-to-left text direction

The component supports both LTR and RTL text directions.

---

### Language Attributes

The component supports language attributes. The `ConvertLanguage` property activates support for language conversion. The `DefaultLanguage` property sets the default language of the document.

# Logging

---

## Configuration of Log Messages

### Usage of the OnLog event handler

To add logging, assign a log event handler to the property OnLog

```
type
  TMyClass = class
  ...
private

  // declare log event handler
  procedure MyLogHandler(Sender: TObject;
                        const logLevel: TLogLevel;
                        const logText: string);

  ...
// assign the log event handler
constructor TMyClass.Create;
begin
  inherited;

  // instantiate ScroogeXHTML component
  SX := ...
  ...
  // set the log handler
  SX.OnLog := MyLogHandler;
end;

// implement logging
TMyClass.MyLogHandler(procedure(Sender: TObject;
  const logLevel: TLogLevel; const logText: string);
begin
  // write string to file, console or outputdebugstring
  ...
end;
```

### Setting the log level

The `LogLevel` property can be used to control the detail level of the logging procedure.

## **Logging over other logging frameworks**

ScroogeXHTML 6.11 introduced support for a logging facade (included in the libraries folder) which can be enabled by defining the conditional symbol

### **SCROOGE\_USE\_SLF4P**

If this symbol is defined, the source folder of the logging facade must be in the project search path. Also, the project (\*.dpr) must include one unit to bind the facade with the actual logging framework.

The logging facade is still in development and only an experimental feature. It is licensed under the Apache 2.0 open source license.

Project home page:

<https://github.com/michaelJustin/slf4p>

## Frequently Asked Questions

---

### Conversion

#### How can I remove the space between lines?

This is the most frequently asked question. HTML has a default paragraph height which will render this RTF

Line 1

Line 2

Line 3

as

Line 1

Line 2

Line 3

#### Solution:

To remove empty space between lines, try to set the **ConvertEmptyParagraph** property to True to replace empty paragraphs, use a Transitional Document Type and define a CSS style for the paragraph element to set the top and bottom margin to 0 pixel:

```
SX.ElementStyles.Values['p'] := 'margin-bottom:0px;margin-top:0px;';
```

The CSS code can also be embedded in the page stylesheet or external stylesheet file:

```
p { margin-bottom:0px;margin-top:0px; }
```

## **How can I remove the indentation?**

The unit `SxConst` contains a global variable `SCROOGE_INDENT_CHAR` with default `' '` which allows to set indentation character in code instead of using `SCROOGE_NO_INDENT`. Set it to an empty character to generate documents without indentation.



# Index

## Reference

AddOuterHTML.....	9, 12, 15f., 19	HyperlinkOptions.....	17
ConvertEmptyParagraph.....	14, 31	IncludeDefaultFontStyle.....	18f.
ConvertHyperlinks.....	16ff.	IPictureAdapter.....	23
ConvertIndent.....	14	IPictureConverter.....	23
ConvertLanguage.....	28	LogLevel.....	29
ConvertPictures.....	22	OnHyperlink.....	17f.
ConvertSpaces.....	14	opFontBGColor.....	12
ConvertUsingPrettyIndents.....	7	opFontColor.....	12, 17f.
DefaultFontColor.....	18f.	opFontHLCOLOR.....	12
DefaultFontName.....	18f.	opFontName.....	12
DefaultFontSize.....	18f.	opFontSize.....	12
DefaultLanguage.....	28	opFontStyle.....	12, 17f.
DocumentType.....	7, 9, 11	OptionsOptimize.....	18f.
ElementClasses.....	20f.	PictureAdapter.....	22f.
ElementStyles.....	20, 31	ReplaceFonts.....	12f.
FontConversionOptions.....	12, 17f.	TabString.....	14
FontSizeScale.....	13	TEmbeddedPicture.....	23
getPictures.....	23	TMemoryPictureAdapter.....	22
hoReplaceEMailLinks.....	17	Unicode.....	5, 28
hoRequireHTTP.....	17	Web Demo.....	25
HyperlinkList.....	17		