

# Getting started with ScroogeXHTML

Michael Justin

*A short guide for the first steps with the RTF to HTML/XHTML converter component*

## Trademarks

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. Oracle and Java are registered trademarks of Oracle and/or its affiliates. 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
Quality Assurance.....	5
ScroogeXHTML for the Java™ Platform.....	5
<b>Installation.....</b>	<b>6</b>
<b>Requirements.....</b>	<b>6</b>
<b>Component Upgrades.....</b>	<b>6</b>
<b>Component installation in Delphi.....</b>	<b>6</b>
Prepared Packages.....	6
<b>Browser 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>Basic 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>14</b>
Conversion of Space Characters.....	14
Tab characters.....	14

<b>Paragraph related Options.....</b>	<b>14</b>
Empty Paragraphs.....	14
Indentation.....	14
<b>Advanced Configuration Options.....</b>	<b>15</b>
<b>Embedding HTML Output in existing Documents.....</b>	<b>15</b>
<b>Hyper Links.....</b>	<b>17</b>
How Hyper Links are detected.....	17
Check List.....	17
Automatic Hyper Link Handling.....	17
E-Mail Links.....	17
HTTP Links.....	18
HyperlinkList.....	18
OnHyperlink Event Handler.....	18
<b>Default Font Styles.....</b>	<b>19</b>
OptionsOptimize Property Group.....	19
<b>ElementStyles and ElementClasses.....</b>	<b>20</b>
List of supported Elements.....	20
ElementStyles.....	21
ElementClasses.....	21
<b>Picture Support.....</b>	<b>22</b>
<b>PictureAdapter.....</b>	<b>22</b>
Purpose.....	22
Example PictureAdapter.....	22
The Init, Write and Finalize Methods.....	22
The PictureHTML Function.....	23
Limitations.....	23
Example implementation TWMFAdapter.....	23
<b>International Support.....</b>	<b>24</b>
<b>Unicode and Code Pages.....</b>	<b>24</b>
Requirements on the Client Side .....	24
Requirements on the Producer Side.....	24

<b>Left-to-right and right-to-left text direction.....</b>	<b>24</b>
<b>Language Attributes.....</b>	<b>24</b>
<b>UTF-8 Encoding.....</b>	<b>25</b>
<b>Description.....</b>	<b>25</b>
<b>Logging and Debugging.....</b>	<b>26</b>
<b>Configuration of Log Messages.....</b>	<b>26</b>
Setting the Logging Detail Level.....	26
<b>Debug Mode.....</b>	<b>26</b>
Activation of Debug Mode.....	26
<b>Frequently Asked Questions.....</b>	<b>27</b>
<b>Conversion.....</b>	<b>27</b>
<b>Index.....</b>	<b>28</b>

# Introduction

---

## About ScroogeXHTML

ScroogeXHTML for Delphi™ is a component which can convert RTF stored in files, strings or a TRichEdit component to [HTML 4.01](#), [HTML 5.0](#) and [XHTML](#). It is fast and easy to customize. Full source code and unlimited upgrade protection are included.

## Features

ScroogeXHTML 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, HTML 5.0.

## Limitations

- the RTF specification contains very many elements and features, ScroogeXHTML does only convert a limited subset. Not supported are for example tables and tabulators
- embedded images may be extracted but will not be converted to other image formats

## Quality Assurance

This component has been developed and tested using

- DUnit unit testing framework (<http://sourceforge.net/projects/dunit/>)
- FastMM4 memory manager and memory leak detector (<http://sourceforge.net/projects/fastmm/>)

## ScroogeXHTML for the Java™ Platform

ScroogeXHTML is also available for the Java(tm) platform.

# Installation

---

## Requirements

ScroogeXHTML supports the following development environments:

- Delphi 6 – 2010
- Free Pascal 2.2 or 2.4

ScroogeXHTML is also available for the Java(tm) platform.

---

## Component Upgrades

If you upgrade from older versions, make a backup of your existing version and make sure that you also have a backup of your own source codes.

If you upgrade from old versions, component properties may have changed and this could cause error messages when you open existing projects with the new version installed.

---

## Component installation in Delphi

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

1. create a new Delphi Package Project
2. add the file ScroogeXHTML\_reg.pas
3. install the package

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

## Prepared Packages

The source code distribution of ScroogeXHTML contains prepared packages for Delphi 6 to 2010 in the *packages* folder for your convenience.

For example, for Delphi 2007 this would be the file *packages\d105\dclScroogeXHTMLD105.dpk*.

## Browser 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.

**ScroogeXHTML1.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'

## 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. 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
<b>RichEditToHTML</b>	this function converts the RTF code in a TRichEdit component to HTML

### 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 ScroogeExample2;
uses
  ScroogeXHTML;
begin
  with TBTScroogeXHTML.Create(nil) do
    try
      WriteLn(Convert('{\rtf1 {\b bold \i Bold Italic \i0 Bold
again}}'));
    finally
      Free;
    end;
  end.
end.
```



The example also shows how to create a component instance at runtime. In some applications it is necessary to create instances of components at runtime - for example, multi-threaded applications such as web servers require one component instance per thread.

The generated XHTML code should look like this:

```
<?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="TBTScroogeXHTML 5.0" />
  </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 be 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.

### Example code

To write an application which can convert a RTF file to a HTML/XHTML file, drop a ScroogeXHTML component on the form, add a button to the form and enter the following code for the OnClick event:

```
var
  RTFFile, XHTMLFile: AnsiString;
begin
  RTFFile := 'c:\windows\desktop\test.rtf';
  XHTMLFile := 'c:\windows\desktop\test.html';
  ScroogeXHTML1.ConvertRTFFile(RTFFile, XHTMLFile);
end;
```

## 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.

## Basic 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
- HTML 5.0

##### XHTML based

- XHTML 1.0 Transitional
- XHTML 1.0 Strict
- XHTML Basic 1.0
- XHTML Mobile 1.0
- XHTML 1.1
- XHTML 2.0

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:

```
...
with TBTScroogeXHTML.Create(nil) do
try
  DocumentType := dtHTML_401_Transitional;
  WriteLn(Convert(EXAMPLE_RTF));
finally
  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 Example4;
uses
  SxMain, SxTypes;
const
  EXAMPLE_RTF = '{\rtf1 {\b bold \i Bold Italic \i0 Bold
again}}';
begin
  with TSxMain.Create(nil) do
    try
      OptionsHead.AddOuterHTML := False;
      FontConversionOptions := [];
      WriteLn(Convert(EXAMPLE_RTF));
    finally
      Free;
    end;
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.

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

---

### 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.

For example 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 2007
    </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 HTML 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;
uses
  SxMain;
begin
  with TSxMain.Create(nil) do
    try
      OptionsHead.AddOuterHTML := False;
      WriteLn(Convert('{\rtf1 {\b bold \i Bold Italic \i0 Bold
again}}'));
    finally
      Free;
    end;
  end.
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.

```
...  
  
ScroogeXHTML1.ConvertHyperlinks := True;  
ScroogeXHTML1.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:', 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.foobar.com';  
  
..
```

## OnHyperlink Event Handler

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

### Example code

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

```
procedure TConverterTest.OnHyperlink(Sender: TObject;  
    var LinkText: WideString);  
begin  
    if LinkText = 'foobar' then  
        LinkText := '<a href="http://www.foobar3.org"  
target="_new">foobar</a>';  
end;  
...  
procedure TConverterTest.Test;  
var  
    ScroogeXHTML1: TBTScroogeXHTML;  
begin  
    ScroogeXHTML1 := TBTScroogeXHTML.Create(nil);  
    ScroogeXHTML1.ConvertHyperlinks := True;  
    ScroogeXHTML1.FontConversionOptions := [opFontStyle,  
opFontColor];  
    ScroogeXHTML1.OnHyperlink := OnHyperlink1;  
    ...  
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:

```
...
ScroogeXHTML1.OptionsOptimize.DefaultFontName := 'Times,serif';
ScroogeXHTML1.OptionsOptimize.DefaultFontColor := '#000000';
ScroogeXHTML1.OptionsOptimize.DefaultFontSize := 10;
ScroogeXHTML1.OptionsOptimize.IncludeDefaultFontStyle := True;
ScroogeXHTML1.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

```
...

ScroogeXHTML1.ElementStyles.Values['p'] := 'foo';
ScroogeXHTML1.ElementStyles.Values['br'] := 'bar';

...
```

## List of supported Elements

The following elements are supported:

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

- ul (bullet list)
- li (list item)

## 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.

```
...  
  
ScroogeXHTML1.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'

```
...  
  
ScroogeXHTML1.ElementClasses.Values['p'] := 'id1';  
  
...
```

The result HTML code for all paragraphs will now be:

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

# Picture Support

---

## PictureAdapter

### Purpose

The ISxPictureAdapter interface allows to create custom picture conversion filters. The component itself can not convert embedded images.

However, it includes an interface which allows to write custom picture converters and connect them to the component. To do this, the application developer writes a subclass of TPictureAdapter (in unit SxPictureSupport) and assigns an instance of this class to the component's property PictureAdapter.

### Example PictureAdapter

Declaration for a custom converter class, which implements the necessary abstract methods Init, Write and Finalize:

#### Code example

```
...  
  
TMyPictureAdapter = class(TPictureAdapter)  
public  
    function PictureHTML: AnsiString; override;  
  
    procedure Init; override;  
    procedure Write(b: byte); override;  
    procedure Finalize; override;  
end;  
  
...
```

### The Init, Write and Finalize Methods

The Init method will be called after the picture type and size have been stored in the PictureAdapter. This method can be used to initialize the custom converter.

The Write method will be called for each picture data byte.

The Finalize method will be called at the end of the picture data and can be use to free objects in the custom adapter which are no longer used.

## The PictureHTML Function

Immediately after the call of the Finalize method, the converter will use the PictureHTML function to get the HTML code for the <img> tag. This function is already declared in the parent class TPictureAdapter.

If you override this class, please verify that the returned HTML string is compatible with the target document type.

## Limitations

- Supported image types EMF, PNG, JPEG, PICT, WMF
- Binary coded pictures (which use the \\bin token) are currently unsupported

## Example implementation TWMFAdapter

The unit ExamplePictureAdapter contains the class TWMFAdapter which extracts WMF image informations and saves them to a file.

## 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.



## UTF-8 Encoding

---

### Description

ScroogeXHTML 5.0 introduced a new option to create UTF-8 encoded documents. Older versions used a numerical representation of Unicode values. This has been the recommended way to encode Unicode characters for older HTML browsers which were not able to process UTF-8 encoded web pages.

The new option uses UTF-8 encoding instead of the numerical representation. To activate UTF-8, compile the application with the switch SCROOGE\_UTF8.

<b>Note</b>	this is an experimental feature, to be used in production environments only after careful testing
-------------	---

# Logging and Debugging

---

## Configuration of Log Messages

### Setting the Logging Detail Level

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

---

## Debug Mode

### Activation of Debug Mode

The debug mode can be enabled with the property DebugMode. It requires that the component has been compiled with the SX\_DEBUG compiler switch.

In debug mode, the HTML code includes all elements of the RTF document - RTF tokens, control characters etc. and the plain text in different colors.

- unknown RTF tokens are red
- known RTF tokens are green
- font names and other unprinted text is silver
- document groups are blue
- document text is black

**Note** In debug mode the converter will create very large HTML documents!

## Frequently Asked Questions

---

### Conversion

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

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

```
ScroogeXHTML1.ElementStyles.Values['p'] := 'margin-  
bottom:0px;margin-top:0px;';
```

#### WingDings characters do not work

Wingdings characters in Web pages is a bad idea, because the font does not use Unicode encoding and is not available on all computers. The intended Wingdings characters may not appear on computers running non-Microsoft operating systems such as Mac OS 9, Mac OS X 10 or Linux. The intended characters are also unlikely to appear on computers that are running Windows when using a standards-compliant browser such as Mozilla, Netscape 7 or Opera 6 or 7. The same problems are found with the Webdings, Wingdings 2 and Wingdings 3 fonts - they should not be used in Web pages.

# Index

## Reference

AddOuterHTML.....	<b>10, 12, 16, 19</b>	HyperlinkList.....	18
ConvertEmptyParagraph.....	14, 27	HyperlinkOptions.....	17f.
ConvertHyperlinks.....	17f.	IncludeDefaultFontStyle.....	19
ConvertIndent.....	14	Linux.....	27
ConvertLanguage.....	24	LogLevel.....	26
ConvertSpaces.....	14	OnHyperlink.....	17f.
ConvertUsingPrettyIndents.....	7	opFontBGColor.....	12
DebugMode.....	26	opFontColor.....	12, 17f.
DefaultFontColor.....	19	opFontHLCOLOR.....	12
DefaultFontName.....	19	opFontName.....	12
DefaultFontSize.....	19	opFontSize.....	12
DefaultLanguage.....	24	opFontStyle.....	12, 17f.
DocumentType.....	7, 9, 11	OptionsOptimize.....	19
ElementClasses.....	20f.	ReplaceFonts.....	12f.
ElementStyles.....	20f., 27	SCROOGE_UTF8.....	25
FontConversionOptions.....	12, 17f.	SX_DEBUG.....	26
FontSizeScale.....	13	TabString.....	14
Free Pascal.....	6	Unicode.....	5, 24, 27
hoReplaceEMailLinks.....	17	UTF-8.....	25
hoRequireHTTP.....	18	WingDings.....	27