SCROOGEXHTML
RTF CONVERTER LIBRARIES

Getting started with

# ScroogeXHTML for the Java™ platform

Version 6.3

**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, service marks, and/or registered trademarks of Embarcadero Technologies, Inc. and are protected by the laws of the United States and other countries. Mac and OS X are trademarks of Apple Inc., registered in the U.S. and other countries. Oracle, WebLogic and Java are registered trademarks of Oracle and/or its affiliates. Other brands and their products are trademarks of their respective holders.

**Licenses**

Roboto Slab font licensed under Apache License, version 2.0

# Contents

# Introduction

## About ScroogeXHTML

### 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.

### Limitations

The library supports a limited subset of the RTF standard. If you are unsure about support for a specific conversion feature, please contact us.

Some of the document elements which will not be converted are:

- Tabulators (a tab character will be replaced by a sequence of non breaking spaces)
- Non-alphabetic characters in the "Symbol" font

### Embedded images

The library extracts raw data of embedded images. The conversion of raw data from WMF or other not web-ready formats to a web-ready format (e. g., PNG or JPG) requires third-party libraries. Habarisoft can not give recommendations for specific graphic libraries.

### API Documentation

The API documentation can be found in the installation folder. A link to the current on-line version can be found on the product home page.

# Installation

## Requirements

ScroogeXHTML for the Java™ platform requires

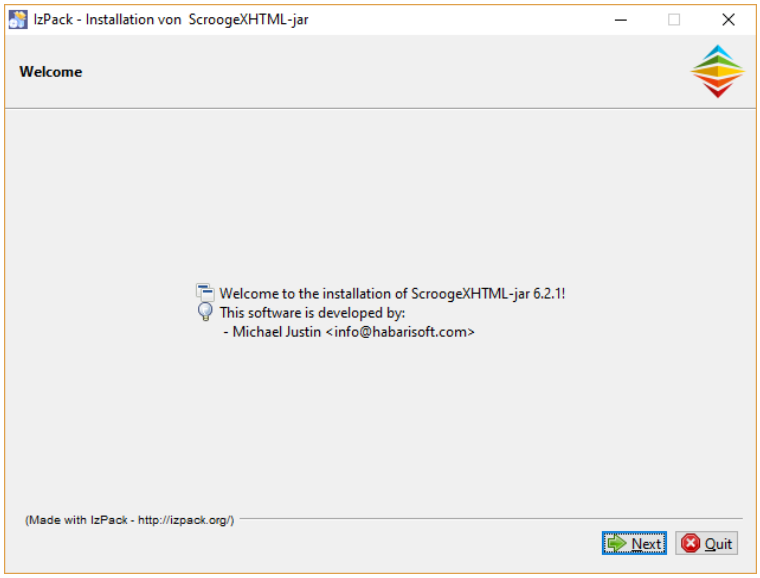
- Java SE 7 (or newer)
- SLF4J (logging framework)
- JDK 7 for development

## Installation steps

The library installer is an executable JAR[1] file created with izPack[2] and works on Microsoft Windows™, Linux™, Solaris™ and Mac OS X™. A Java Run-time Environment is required to execute it.

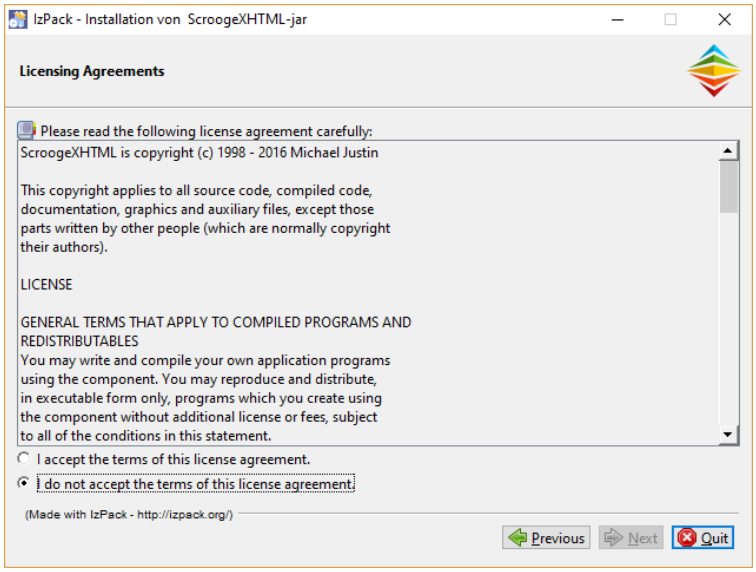
To launch the installer, double-click it. The installer will guide you through the installation steps.

The installation begins with a language selection dialog.

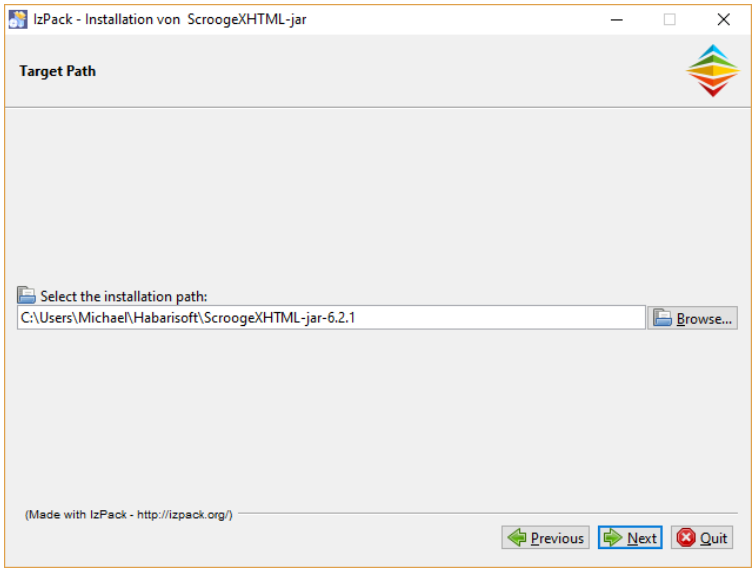



Welcome Page

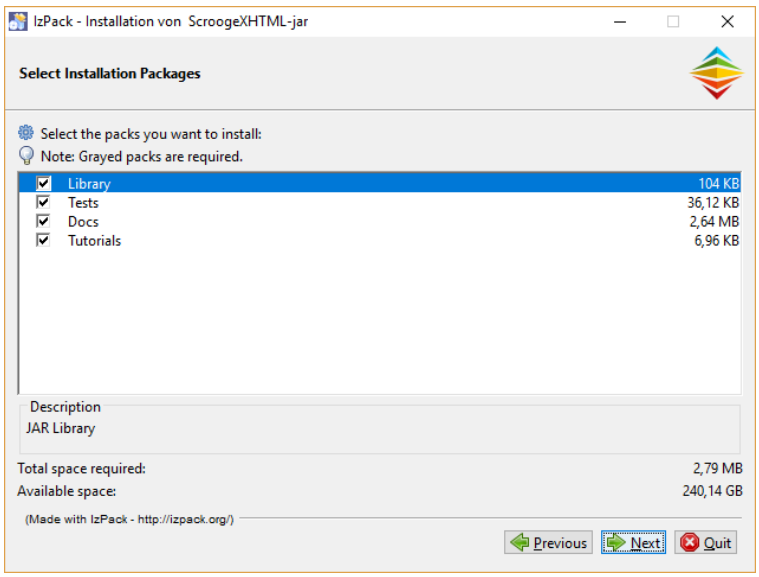
1 https://docs.oracle.com/javase/7/docs/technotes/guides/jar/jarGuide.html
2 http://izpack.org/




Licensing Agreements

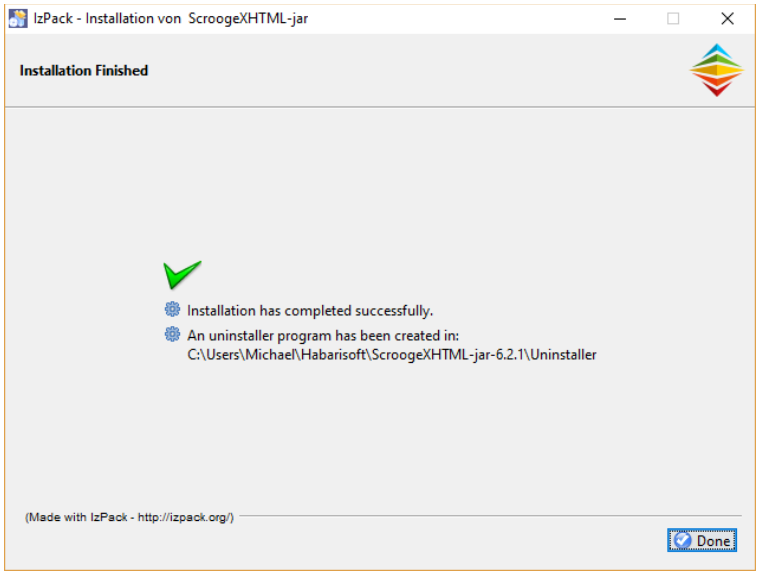



Target Path




Select Installation Packages

(The list of available installation packages depends on the license type)




Installation finished

Depending on your choice, an uninstaller will be installed in a sub-directory of the installation folder.

## Maven dependency

Maven

```
<dependencies>
  ...
  <dependency>
    <groupId>com.habarisoft</groupId>
    <artifactId>ScroogeXHTML</artifactId>
    <version>6.3.0</version>
  </dependency>
  ...
</dependencies>
```

## Gradle dependency

Gradle

```
dependencies {
    // ... other dependencies here
    compile 'com.habarisoft:ScroogeXHTML:6.3.0'
}
```

# Tutorial one: simple conversion

## What it does

This example converts a hard-coded RTF document to a HTML5 document named 'tutorial-1.html' in the current directory.

It sets the `AddOuterHTML` property which causes generation of surrounding HTML head and body code. The converted HTML is inserted within the body of the document.

### Java code

Code example

```
public class Tutorial1 {

    public static final void main(String[] args) throws IOException {

        String rtf = "{\\rtf1 {\\b bold \\i Bold Italic \\i0 Bold
again} \\par}";

        // create a converter instance
        ScroogeXHTML scrooge = new ScroogeXHTML();

        // configure conversion options
        scrooge.setAddOuterHTML(true);

        // convert RTF and write HTML to file
        scrooge.convert(rtf, new File("tutorial-1.html"));

    }
}
```

Compile and run this class, and open the result document in a web browser or a text editor.

### Result HTML

HTML

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Untitled document</title>
    <meta content="ScroogeXHTML for the Java(tm) platform 6.0"
name="generator">
  </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>
```

# Tutorial two: additional HTML code

## What it does

This example converts a hard-coded RTF document to a HTML5 document named 'tutorial-2.html' in the current directory.

It shows how a HTML fragment generated by the converter can be embedded in other HTML code and then saved to a file.

## Java code

Code example

```
public class Tutorial2 {

    public static final void main(String[] args) throws IOException {

        String rtf = "{\\rtf1 Hello {\\b World} from ScroogeXHTML \\par}";

        // create a converter instance
        ScroogeXHTML scrooge = new ScroogeXHTML();

        // convert RTF and store HTML in String variable
        String converted = scrooge.convert(rtf);

        // wrap with required HTML5 elements
        String html = "<!DOCTYPE html>\n"
                + "<html>\n"
                + "  <head>\n"
                + "    <title>\n"
                + "      Untitled document\n"
                + "    </title>\n"
                + "    <meta http-equiv=\"content-type\"
content=\"text/html; charset=UTF-8\">\n"
                + "  </head>\n"
                + "  <body>\n"
                + "    <p>additional paragraph before</p>\n"
                + converted
                + "    <p>additional paragraph after</p>\n"
                + "  </body>\n"
                + "</html>";

        try {
            writeHtmlFile(html);
        } catch (IOException ex) {
```

```
            Logger.getLogger(Tutorial2.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private static void writeHtmlFile(String html) throws IOException {
        OutputStream os = new FileOutputStream(new File("tutorial-
2.html"));
        Writer writer = new OutputStreamWriter(os,
StandardCharsets.UTF_8);
        try (BufferedWriter outWriter = new BufferedWriter(writer
        )) {
            outWriter.write(html);
        }
    }
}
```

Compile and run this class, and open the result document in a web browser or a text editor.

## Result HTML

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Untitled document
    </title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <p>additional paragraph before</p>
  <p>Hello <span style="font-weight:bold;">World</span> from ScroogeXHTML
</p>
    <p>additional paragraph after</p>
  </body>
</html>
```

# Configuration

## **Embedding HTML:** Usage of AddOuterHTML

If you convert RTF using the methods `ScroogeXHTML#convert(String rtf)`, `ScroogeXHTML#convert(String rtf, Charset charset)` or `ScroogeXHTML#String convert(final ByteArrayInputStream rtf)`, the converter by default returns only a HTML fragment for the RTF input, without enclosing it in `<html>…<body>…</body></html>` tags.

This HTML fragment then can be used in a larger HTML document. The application code is responsible for adding all required HTML elements to complete the document.

Choosing the correct Charset and document type (HTML5 or XHTML) for the result document is also important. Note that it is highly recommended to specify the result document Charset whenever you save the HTML to a file, or write it to a HTTP response, to avoid encoding problems on the receiver side.

The property `AddOuterHTML` controls whether the enclosing HTML will be generated by the converter. Use `setAddOuterHTML(true)` to switch it on.

For conversions to files, the `AddOuterHTML` property must always be set to `true`. If the property is `false`, the converter will throw a `UnsupportedOperationException`.[3]

3 This is a breaking change in version 6.0

# **Hypertext support:** Conversion of Hyperlinks

For speed and security reasons, the converter does not convert hyperlinks in the RTF document to clickable hyperlinks by default. There are two levels of hyperlink support in ScroogeXHTML for the Java™ platform. The first uses simple text attribute based conversion, the second uses hidden RTF fields.

## Attribute-based hyperlink detection

If hyperlink conversion is enabled with `setConvertHyperlinks(true)`, the converter will convert blue and underlined text fragments to hyperlinks, using the formatted text as the target address and also as the hyperlink display text. For example, http://example.com will be converted to

```
<a href="http://example.com">http://example.com</a>
```

Note: the converter is able to detect this hyperlink style only if the properties for font color and font style conversion, `ConvertFontColor` and `ConvertFontStyle`, are set to `true` (this their default value).

## Hyperlink field detection

Many RTF documents use specific hidden fields to store the Hyperlink target and the corresponding display text.

To enable hyperlink conversion of these RTF hyperlink fields, in addition to `setConvertHyperlinks(true)also` use `setConvertFields(true)`.

If a hidden field does not specify a hyperlink, the converter will only insert the display text (the 'result value' of the hidden field) in the output document.

## Hyperlink support configuration matrix

| **Property settings if you need...** | **ConvertHyperlinks** | **ConvertFields** |
|---|---|---|
| - no hyperlink conversion | `false` | - |
| - only conversion for blue and underlined text | `true` | `false` |
| - conversion for blue and underlined text, and for hyperlink fields | `true` | `true` |

Hint: to convert only hyperlink fields but ignore no blue and underlined text, first set the properties `ConvertHyperlinks` and `ConvertFields` to `true` and then set the property `ConvertHyperlinksForBlueUnderlinedText` to `false`.

# Table conversion

ScroogeXHTML for the Java™ platform supports conversion of simple RTF tables to HTML.

The converter does not convert tables by default. Any tables in the RTF input document will be converted to text paragraphs.

Table conversion is activated with `setConvertTables(true)`.

Because RTF document writers can create highly complex RTF table code, conversion results may not be perfect.

# **Size Optimization:** Default Font Properties

Document size can be optimized with the usage of CSS for frequently used font properties which can be set using the `DefaultFontSize`, `DefaultFontName` and `DefaultFontColor` properties.

Setting the `IncludeDefaultFontStyle` property to true then has these effects:

- if `AddOuterHTML` is true, the HTML head section will contain a CSS definition for the default font style
- the converter will create font style attributes only for text parts which differ from the values of the `DefaultFontSize`, `DefaultFontName` and `DefaultFontColor` properties

## Example

If most text in the document uses "Arial, 14 pt, black", set the `DefaultFontSize`, `DefaultFontName` and `DefaultFontColor` properties to these values, and set `IncludeDefaultFontStyle` to true.

If the document is converted with AddOuterHTML to true, the HTML head section will contain the following CSS definition:

Code example

```
<style type="text/css">
  <!-- BODY
  {font-family:Arial,sans-serif;font-size:14pt;color:#000000; }
  -->
 </style>
```

# **Picture Extraction:** PictureAdapter Interface

Picture extraction is activated by assigning a `PictureAdapter` implementation and `setConvertPictures(true)`.

The library includes basic implementations of the `PictureAdapter` interface.

## MemoryPictureAdapter

`MemoryPictureAdapter` keeps all extracted picture data in memory and returns hyperlinks to HTTP picture resources, which are then inserted in the result document.

This implementation is useful for web server environments where the server returns the image data back to the client. In the most simple implementation, the server keeps the image data in memory for the duration of a client session, and returns the image data dynamically when the browser requests the image resource URLs. Of course this requires HTTP session management and sufficient memory.

**Example for a link element:**

Code example

```
<img alt="picture" height="132" width="156" src="image1.png" />
```

The image URLs will be numbered automatically.

The class allows to set a base path with `setBase(String base)`, for example `scrooge.setBase("/images/")`, so that the result URL will be `"/images/image1.png"`.

## MemoryPictureAdapterBase64

`MemoryPictureAdapterBase64` extends `MemoryPictureAdapter` but returns Image Data URIs for pictures which do not exceed a given maximum size. For larger images, it will return the external image URL as defined by its super class.

By default, the size threshold is set to 32 kB. The threshold can be set with the `maxSize` constructor argument.

Data URIs are fully supported by most major browsers, and partially supported in Internet Explorer and Microsoft Edge.

Code example

```
scrooge = new ScroogeXHTML();
scrooge.setConvertPictures(true);
PictureAdapter adapter = new MemoryPictureAdapterBase64();
scrooge.setPictureAdapter(adapter);
```

**Example for a Data URI link:**

Code example

```
<img alt="picture" height="132" width="156"
src="… />
```

# **Post Processing:** manipulation of the result DOM

## The PostProcessListener interface

The converter stores a list of post progress listeners in its `PostProcessListener` property. The listeners implement the `PostProcessListener` interface, which has one method, `postProcess`.

The converter passes an instance of `PostProcessEventObject` to the postProcess method. This event object carries references to the converter and the `org.w3c.dom.Document` instance with the result DOM.

## Tutorial 3: fix missing http:// in hyperlinks

Tutorial 3 converts a RTF documents which contains a simple (blue and underlined formatted) hyperlink.

The hyperlink text does not begin with a valid protocol name such as `http://` and this causes a non functional hyperlink in the result HTML:

HTML

```
<p>
  <a href="example.com">example.com</a>
</p>
```

To fix this, we want to apply post processing code which modifies all `<a>` elements so that they begin with `http://`

The result should be:

HTML

```
<p>
  <a href="http://example.com">example.com</a>
</p>
```

Our solution will use the XPath expression `//a[not(contains(@href, '://'))]` to find all `<a>` elements in the document whose `href` attribute do not contain the character sequence "://".

For all found elements, our code then inserts "http://" in the value of the `href` attribute.

Notes

- this is pure demonstration code
- there is no guarantee that the result `href` value will be a valid internet address

## Source code

The following source code example shows the `PostProcessListener` implementation and its `postProcess` method. The full source code can be found in the `tutorials` package of the library.

Code example

```
// create a converter instance
ScroogeXHTML scrooge = new ScroogeXHTML();

// we want simple HTML output for this example
scrooge.setConvertFontSize(false);
scrooge.setConvertFontName(false);

// enable hyperlink conversion
scrooge.setConvertHyperlinks(true);

// add post process listener
scrooge.getPostProcessListeners().add(new PostProcessListener() {
  @Override
  public void postProcess(PostProcessEventObject e) {
      try {
        XPathFactory xpathFactory = XPathFactory.newInstance();
        // XPath to find hyperlink nodes.
        XPathExpression xpathExp = xpathFactory.newXPath().compile(
          "//a[not(contains(@href, '://'))]");
        NodeList links = (NodeList) xpathExp.evaluate(e.getDocument(),
XPathConstants.NODESET);
        for (int i = 0; i < links.getLength(); i++) {
          Element a = (Element) links.item(i);
          String href = a.getAttribute("href");
          a.setAttribute("href", "http://" + href);
        }
      } catch (XPathExpressionException ex) {
       Logger.getLogger(Tutorial3.class.getName()).log(Level.SEVERE, null,
ex);
    }
  }
});

// convert RTF to HTML
String html = scrooge.convert(rtf);
```

# New in 6.0

## Improved RTF table conversion

- **table borders**: if the first table cell is bordered, the whole table will be rendered cell borders
- **left margin**
- **table and column widths**: column widths will be converted, the output HTML uses colgroup elements which specify the column widths in pixel. The total table width is included in the table header.

## Embedded images

The traditional `MemoryPictureAdapter` class in ScroogeXHTML for the Java™ platform generates image link elements which point to a resource location `<img src="…">`. This keeps the document small, but requires making the image resources accessible for the web browser at the given location.

In some cases however, it is useful to embed the whole image in-line in the web page as if they were external resources.

The new `MemoryPictureAdapterBase64` class returns Data URIs for small JPEG and PNG images. By default, the size threshold is set to 32 kB.

Code example

```
scrooge = new ScroogeXHTML();
scrooge.setConvertPictures(true);
PictureAdapter adapter = new MemoryPictureAdapterBase64();
scrooge.setPictureAdapter(adapter);
```

The new class inherits from `MemoryPictureAdapter`, and will return the inherited result for images which exceed the size limit.

Data URIs are fully supported by most major browsers, and partially supported in Internet Explorer and Microsoft Edge.

## Event handlers for DOM post processing

The converter internally uses an XML DOM tree to create the HTML document structure. Before converting the DOM to the result HTML5 String, the converter calls a sequence of post processing handlers, which apply optimizations and custom modifications on the DOM tree. Post processing handlers must implement the `PostProcessListener` interface.

The converter stores the event handlers in its `PostProcessListeners` property which is a list of `PostProcessListener` implementations. By default, the converter library creates and assigns post process handlers to perform these tasks

- strip empty (white space-only) text nodes
- strip empty span nodes
- strip attribute-less span nodes
- replace empty paragraph (<p>) nodes with <br> nodes

These default PostProcessListener implementations are located in the `com.habarisoft.scroogexhtml.tidy` package.

Application code may create and add more post process listeners as needed.

## Export to HTML5 or XHTML 1.0

HTML5 and XHTML 1.0 are supported. The document type can be selected with setDocumentType:

Code example

```
setDocumentType(DocumentType.HTML5);
```

or

Code example

```
setDocumentType(DocumentType.XHTML);
```

## IOException in method signatures

The signatures of conversion methods now include throws `IOException` to provide error information and custom exception handling in client code.

# Frequently Asked Questions

## General

### Is there a trial version of the library?

A trial version download is not available. To check if the library meets your requirements, you can try the on-line demo or purchase a Single Developer license, which includes a 14 days full money back guarantee. This allows to test the full version of the library without any risk. The reseller (ShareIt) will give a full refund if you find that the library does not work as expected.

### Where can I download updates of the library?

The library home page contains a link to the download area for registered users. Habarisoft will send you the download area credentials (user name and password) when a new release is available.

## Licensing

### Is your license on a per-developer basis?

Yes, each developer that uses our products must have their own license.

### Does the license expire?

No, the licenses are perpetual. However, you will be using the last product version released before your free upgrade period expired.

## Server Deployment license

### When are Server Deployment licenses required?

Server Deployment Licenses are required if ScroogeXHTML for the Java™ platform is used on the server side of a client/server application.

For more details, please check the online FAQ at

https://www.scroogexhtml.com/#faq,

and the license type page at

https://www.scroogexhtml.com/scroogexhtml_license.html

# Installation

## IDE integration in Maven projects

**For NetBeans IDE:**

1. In Maven project open "Add dependency" dialog
2. Make up some groupId, artifactId and version and fill them, OK
3. Dependency will be added to the pom.xml and will appear under "Libraries" node of maven project
4. Right-click Lib node and "manually install artifact", fill the path to the jar

The Jar should be installed to local Maven repository with coordinates entered in step 2

**For Maven (command line):**

http://maven.apache.org/guides/mini/guide-3rd-party-jars-local.html

# Picture support

## Does the library convert embedded pictures to web-ready images?

No, the library does not convert embedded pictures in general. It extracts binary picture data from the RTF document.

The picture data may be in WMF, JPEG, or other formats. The conversion of raw data from WMF or other not web-ready formats to a web-ready format (e. g., PNG or JPG) requires third-party libraries.

Habarisoft can not give recommendations for specific graphic libraries.

## Data URI image embedding

Version 6.0 introduced limited support for Data URI image embedding.

# Can I use the library on Android?

Yes, starting with version 5.3, it supports the Android platform.

# Conversion

## Why are empty paragraphs not in the result page?

Many HTML browsers do not show `<p>` elements when they do not contain any text.

Example:

Line 1

Line 2

Line 3

will look different in the HTML browser

Line 1

Line 2

Line 3

You can set the `ConvertEmptyParagraphs` property to true for most document types (except their "strict" variations).

The result HTML then will contain `<br>` or `<br />` instead of empty `<p>` elements, and look as expected.

## How can I remove the space between lines?

HTML browsers use default paragraph styles, which will render paragraphs in RTF documents with a bigger space between lines than RTF editors.

For example a RTF document which appears like this

Line 1

Line 2

Line 3

will look different in the converted HTML document

Line 1

Line 2

Line 3

### Solution:

To remove empty space between lines, define a CSS style for the paragraph element, which sets the margins to 0:

CSS

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

# Web Applications

## Why is the indentation missing of I use the converter in a web application?

The conversion requires Xerces/Xalan classes which are not included in the classpath.

As a work around, add Xalan to the web application.

If you are using Maven, add this dependency:

Maven

```
        <dependency>
            <groupId>xalan</groupId>
            <artifactId>xalan</artifactId>
            <version>2.7.0</version>
            <type>jar</type>
        </dependency>
```

# ScroogeXHTML for Delphi and Free Pascal

ScroogeXHTML is also available for Delphi and Free Pascal.

# Index

## Reference