SCROOGEXHTML
RTF CONVERTER LIBRARIES

Getting started with

ScroogeXHTML for the Java™ platform

Version 9.3

This documentation is work in progress

LIMITED WARRANTY

No warranty of any sort, expressed or implied, is provided in connection with the library, including, but not limited to, implied warranties of merchantability or fitness for a particular purpose. Any cost, loss or damage of any sort incurred owing to the malfunction or misuse of the library or the inaccuracy of the documentation or connected with the library in any other way whatsoever is solely the responsibility of the person who incurred the cost, loss or damage. Furthermore, any illegal use of the library is solely the responsibility of the person committing the illegal act.

By using this program you accept these responsibilities, and give up any right to seek any damages against the authors in connection with this program.

Specifications subject to change without notice.

Contents

Installation

Requirements

ScroogeXHTML for the Java™ platform minimum requirements:

- JDK 8 for development
- Java SE 8 at run-time
- SLF4J – Simple Logging Facade for Java

Installation

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.

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

Directory structure

Source Edition

```
<inst>
\- addons                                        Add-ons
   \- ...
\- apidocs                                       JavaDoc documentation
   \- ...
\- docs
   \- ScroogeXHTMLGettingStarted.pdf             This document
\- examples                                      Examples and tutorials
   \- ...
\- src
  \- main                                        Library source code
  \- test                                        Test source code
\- pictures                                      Picture support code
   \- ...
\- Uninstaller
   \- uninstaller.jar
.installationinformation
license.txt                                  License information
ScroogeXHTML-9.3.0.jar                       Precompiled library
ScroogeXHTML-9.3.0-javadoc.jar               Compressed JavaDoc
ScroogeXHTML-9.3.0-sources.jar               Compressed source code
```

Jar Edition

```
<inst>
\- addons                                        Add-ons
   \- ...
\- apidocs                                       JavaDoc documentation
   \- ...
\- docs
   \- ScroogeXHTMLGettingStarted.pdf             This document
\- examples                                      Examples and tutorials
   \- ...
\- pictures                                      Picture support code
   \- ...
\- Uninstaller
   \- uninstaller.jar
.installationinformation
license.txt                                  License information
ScroogeXHTML-9.3.0.jar                       Precompiled library
ScroogeXHTML-9.3.0-javadoc.jar               Compressed JavaDoc
```

Maven and Gradle

If you install the library in a repository manager, you may access it using these coordinates:

Maven

```
<dependency>
  <groupId>com.scroogexhtml</groupId>
  <artifactId>ScroogeXHTML</artifactId>
  <version>9.3.0</version>
</dependency>
```

Gradle

```
  implementation 'com.scroogexhtml:ScroogeXHTML:9.3.0'
```

Redistribution

The library (ScroogeXHTML-9.3.0.jar) must not be included with your software as a separate file. For redistribution, integrate it within your application Jar to avoid unauthorized redistribution.

Tutorials

Tutorial one: "Hello World!"

This example converts a RTF document to HTML5. It sets the `AddOuterHTML` property to `true` which causes generation of surrounding HTML head and body code.

`Java code`

```
import com.scroogexhtml.ScroogeXHTML;

import java.io.IOException;

public class Tutorial1 {

    public static void main(String[] args) throws IOException {
        ScroogeXHTML scrooge = new ScroogeXHTML();
        scrooge.setAddOuterHTML(true);
        String html = scrooge.convert("{\\rtf1 \"Hello world!\"}");
        System.out.println(html);
    }
}
```

`Output`

```
<!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 9.3.0"
name="generator">
  </head>
  <body>
    <p>"Hello world!"</p>
  </body>
</html>
```

The full source code is included in examples/src/main/java/com/scroogexhtml/tutorials

Tutorial two: HTML fragments

This example converts two RTF documents to HTML5 fragments and concatenates them. It does not set the `AddOuterHTML` property to `true`, so the conversion will not generate surrounding HTML code. Instead, the surrounding HTML code is created by the program.

Java code

```
public class Tutorial2 {

    public static void main(String... args) {

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

        // convert RTF to HTML
        String fragment1 = converter.convert("{\\rtf1 first line\\par
second line}");
        String fragment2 = converter.convert("{\\rtf1 third line\\par last
line}");

        // concatenate
        System.out.println(
                "<html>\n"
                + "  <!-- first fragment -->\n"
                + fragment1
                + "  <!-- second fragment -->\n"
                + fragment2
                + "</html>");
    }
}
```

Output

```
<html>
  <!-- first fragment -->
  <p>first line</p>
  <p>second line</p>
  <!-- second fragment -->
  <p>third line</p>
  <p>last line</p>
</html>
```

The full source code is included in examples/src/main/java/com/scroogexhtml/tutorials

Embedding HTML

Usage of AddOuterHTML

If you convert RTF using the methods

- `void convert(String rtf)`
- `void convert(String rtf, Charset charset)`
- `String convert(final ByteArrayInputStream rtf)`

the converter by default returns only the content of the document **body element**, without enclosing it in `<html>…<body>…</body></html>` tags. This fragment can be used in a larger document.

Code example

```
<p>
  This is a HTML fragment, it is not embedded in outer HTML.
</p>
```

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

Note	For conversions to files, the AddOuterHTML property must always be set to true. If the property is false, the converter will throw an exception.

Character encoding and document type

Choosing the correct charset[3] and document type (HTML5 or XHTML) for the result document is important.

Note	Always specify the result document charset when you save the HTML to a file or write it to a HTTP response to avoid encoding problems on the receiver side

3 https://docs.oracle.com/javase/7/docs/api/java/nio/charset/Charset.html

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` set to true, the HTML head section will contain the following CSS definition:

Code example

```
<style>
  <!--generated styles-->
  body {font-family:Arial,sans-serif;font-size:14pt;color:#000000}
</style>
```

Font name substitution

Default font replacing

The converter by default replaces font names using the class `DefaultFontReplacer`.

It uses a list of font names which should be replaced. Replacing is done if the font in the RTF starts with a name in this list. For example, for font names starting with “Times”, the converter uses “Times,serif”, and for font names starting with “Courier”, it will use “Courier,monospace”.

To add or modify this list, use the property `ReplaceFonts`. For example:

Code example

```
converter.getReplaceFonts().put("Arial", "Arial New");
```

Custom font replacing

It is possible to assign a custom implementation of the `FontReplacing` interface, which will be used instead of the default. The following code example will convert any font name to “Comic Sans”:

Code example

```
converter.setFontReplacing((c, name, family) -> "Comic Sans");
```

Table conversion

Overview

The converter supports conversion of **simple** RTF tables to HTML. The library does not convert tables by default. By default, tables in the RTF input document will be converted to text paragraphs.

To enable conversion of tables, set the ConvertTables property to true.

Code example

```
converter.setConvertTables(true);
```

Notes	As some RTF document writers create highly complex RTF code for tables, conversion results may not be perfect. Enabling table support also enables the experimental option EXPERIMENTAL_CONVERT_TABLE_BORDERS.

Hypertext support

Overview

Hyperlink field detection

Most 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

```
converter.setConvertHyperlinks(true)
```

also use

```
converter.setConvertFields(true)
```

Note	If a hidden field does not specify a hyperlink, the converter will only insert the "display text" of the hidden field in the output document

Differences between RTF and HTML tables

Vertical cell alignment

In HTML, the default value for vertical alignment of table cell content is `MIDDLE`.[4]

4 https://stackoverflow.com/questions/33487148/

The converter assumes that RTF table cell content should be aligned with the top of the cell by default. To keep the output document size small, the conversion adds style elements only for cells which are aligned vertically or with the bottom of the cell.

In order to apply the vertical alignment as the default for table cells, the CSS code `td {vertical-align: top}` must be included.

Code example

```
scrooge.setStyleSheetInclude("td {vertical-align: top;}");
```

Picture data extraction

PictureAdapter Interface

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

Code example

```
scrooge = new ScroogeXHTML();

scrooge.setConvertPictures(true);

PictureAdapter adapter = new MyPictureAdapter();

scrooge.setPictureAdapter(adapter);

// run the conversion
...
```

Example code

See Picture conversion support on page 25.

Language support

How to set the lang Attribute

The example shown below sets the default language of the document to “en” (English). In the result document the `lang` attribute on the html element has the value “en”.

Code example

```
scrooge = new ScroogeXHTML();
scrooge.setDefaultLanguage("en");
```

Why you should use the `lang` attribute on the `<html>` element

By using `lang`, you get the benefits of hyphen support in your (modern) browser that you otherwise would not get (assuming you use `hyphens:auto` in your CSS).[5],[6]

Add `hyphens:auto` in your CSS:

Code example

```
scrooge.setStyleSheetInclude("p {hyphens: auto;}");
```

5 https://adrianroselli.com/2015/01/on-use-of-lang-attribute.html
6 https://www.w3.org/International/questions/qa-lang-why

Post processing

Overview: manipulation of the result DOM tree

Technical background

The converter internally uses an XML DOM tree to create the HTML document structure. Before converting the DOM to the result HTML5 or XHTML 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.

PostProcessListeners property

The converter stores the event handlers in its `PostProcessListeners` property which is a list of `PostProcessListener` implementations.

Performance

Post processing may cause a significant increase of the conversion time.

Example

Code example

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

// create and add a post processor
PostProcessListener listener = new PostProcessListenerExample();

scrooge.getPostProcessListeners().add(listener);

// run the conversion
...
```

No post processing by default

The converter creates and adds no post process handlers by default (on creation).

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.

Example: add elements to the HTML head section

This example shows how a post process listener can be used to add a meta date element to the HTML head.

Code example

```
public class Main {

    public static void main(String... args) {
        ScroogeXHTML scrooge = new ScroogeXHTML();
        scrooge.setAddOuterHTML(true);

        scrooge.addPostProcessListener(new PostProcessListener() {
            @Override
            public void postProcess(PostProcessEventObject e) {
                Document doc = e.getDocument();
                Element html = doc.getDocumentElement();
                Node head = html.getFirstChild();

                // add meta date
                Element metaDate = doc.createElement("meta");
                metaDate.setAttribute("name", "date");
                metaDate.setAttribute("content", new Date().toString());
                head.appendChild(metaDate);
            }
        });

        String html = scrooge.convert("{\\rtf1 Hello world}");
        System.out.println(html);
    }
}
```

Example: fix missing https:// in hyperlinks

The hyperlink text does not begin with a valid protocol name such as `https://` 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 `https://`

The result should be:

HTML

```
<p>
  <a href="https://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 "`https://`" 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 is included in the examples folder (Tutorial3.java in package `com.scroogexhtml.tutorials`).

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.setConvertFields(true);
        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", "https://" + href);
                    }
                } catch (XPathExpressionException ex) {

Logger.getLogger(Tutorial3.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });

        // convert RTF to HTML
        scrooge.setAddOuterHTML(true);
        scrooge.convert(rtf, Paths.get("tutorial-3.html"));
```

Picture conversion support

Complimentary Support Classes

Support notice

All provided example implementations of the PictureAdapter interface are complimentary example code. They are work in progress, not optimized for performance or memory usage and not fully covered by tests.

Example implementations of the PictureAdapter are included in the Pictures source folder.

Class	Picture formats	
MemoryPictureAdapter	JPEG, PNG	
MemoryPictureAdapterDataURI	JPEG, PNG, BMP, WMF	

MemoryPictureAdapter

`MemoryPictureAdapter` keeps all extracted picture data in memory and creates 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. 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" />
```

Picture naming

The image URLs will be named and numbered automatically.

Picture base path

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

Code example

```
<img alt="picture" height="132" width="24" src="/images?image1.png" />
```

MemoryPictureAdapterDataURI

`MemoryPictureAdapterDataURI` embeds Image Data URIs in the document for pictures which do not exceed a defined maximum size.

For larger images, it will embed an external image link.

Code example

```
scrooge = new ScroogeXHTML();

scrooge.setConvertPictures(true);

PictureAdapter adapter = new MemoryPictureAdapterDataURI();

scrooge.setPictureAdapter(adapter);

// run the conversion
...
```

Example for a Data URI link:

Code example

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

Data URI compatibility notes

Data URIs are fully supported by major browsers.[7] **However, browsers are not required to support any particular maximum length of data**.[8]

Data URIs are not separately cached from their containing documents (e. g. CSS or HTML files) so data is downloaded every time the containing documents are re-downloaded.[9]

Recommendation: use Data URIs only for moderate image sizes.

Picture naming

The image URLs will be named and numbered automatically.

Picture base path

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

7 https://caniuse.com/datauri
8 https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URIs
9 https://stackoverflow.com/questions/4791807/

Advanced conversion options

Overview

Advanced conversion options are not controlled by individual properties, but by setting entries in a list of flags, using the `setOutputProperty` method.

Code example

```
scrooge.setOutputProperty(<property key>, <flag>);
```

This table lists available conversion options, their valid values, defaults, and which area of the conversion they affect.

Key	Values	Area
CONVERT_PARAGRAPH_BORDERS	yes | **no**	Paragraphs

Details

CONVERT_PARAGRAPH_BORDERS

This conversion option enables support for paragraphs which are formatted with a simple border box; the value must be “yes” or “no”.

Code example

```
// enable paragraph border conversion
scrooge.setOutputProperty(ConversionKeys.CONVERT_PARAGRAPH_BORDERS,
"yes");
```

Merging

If consecutive paragraphs use the same border style, they may be merged with a post processor. An example post processor removes all identical style nodes between paragraphs and replaces them with one parent div node.

Code example

```
scrooge.addPostProcessListener(new MergeBorderDivNodes());
```

Experimental conversion options

Overview

Advanced conversion options are not controlled by individual properties, but by setting entries in a list of flags, using the `setOutputProperty` method.

Code example

```
scrooge.setOutputProperty(<property key>, <flag>);
```

This table lists experimental conversion options, their valid values, defaults, and which area of the conversion they affect.

Key	Values	Area
EXPERIMENTAL_CONVERT_HEADERS_AND_FOOTERS	yes | **no**	Text
EXPERIMENTAL_SUPPORT_LIST_TABLE	yes | **no**	Lists
EXPERIMENTAL_SUPPORT_STAR_PN	yes | **no**	Lists
EXPERIMENTAL_SUPPORT_MULTILEVEL	yes | **no**	Lists
EXPERIMENTAL_CONVERT_TABLE_BORDERS	yes | **no**	Tables

Details

EXPERIMENTAL_CONVERT_HEADERS_AND_FOOTERS

This conversion option enables support for **header and footer text** conversion.

By default header and footer will not appear in the output document; the value must be "yes" or "no".

Note	This option is experimental and unsupported. The converter will emit multiple header and footer tags in the result document if there is more than one RTF header or footer definition.

EXPERIMENTAL_SUPPORT_LIST_TABLE

This conversion option enables support for the **RTF list table** (Word 97); the value must be "yes" or "no".

The list table is a section in the RTF header which stores styles for numbered and unnumbered lists. The document then refers to a specific style by its id.

Note	This option is experimental and unsupported because some RTF writers generate malformed list tables. Use with caution.

EXPERIMENTAL_SUPPORT_STAR_PN

This conversion option enables support for list formatting based on `\*\pn` RTF tokens (Word 6.0/95 RTF); the value must be "yes" or "no".

Note	This option is experimental and unsupported. Use with caution.

EXPERIMENTAL_SUPPORT_MULTILEVEL

This conversion option enables **support for multilevel numbered and unnumbered lists** in conjunction with `EXPERIMENTAL_SUPPORT_LIST_TABLE` (Word 97); the value must be "yes" or "no".

Note	This option is experimental and unsupported because some RTF writers generate malformed list tables. Use with caution.

Code example

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

// enable experimental list table support
scrooge.setOutputProperty(ConversionKeys.EXPERIMENTAL_SUPPORT_LIST_TABLE,
"yes");

// enable experimental multi-level support
scrooge.setOutputProperty(ConversionKeys.EXPERIMENTAL_SUPPORT_MULTILEVEL,
"yes");

// run the conversion
```

EXPERIMENTAL_CONVERT_TABLE_BORDERS

This conversion option enables support for **table border formatting**; the value must be "yes" or "no".

Enabling table support by SetConvertTables(true) enables this feature. If you want to disable it, set the option to "no" after enabling table support.

Code example

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

// enable table conversion
converter.setConvertTables(true);

// disable border formatting support
converter.setOutputProperty(ConversionKeys.EXPERIMENTAL_CONVERT_TABLE_BORD
ERS, "no");
```

Experimental feature notice

- If you use experimental, be aware that not all RTF writers generate correct or conforming code
- Experimental features may be removed or changed significantly in future releases

Frequently Asked Questions

Which CSS entries should I consider?

Code example

```
scrooge.setStyleSheetInclude("body, p {\n"
            + "  margin: 0px;\n"
            + "}\n"
            + "table {\n"
            + "  border-collapse: collapse;\n"
            + "}\n"
            + "td {\n"
            + "  vertical-align: top;\n"
            + "}\n");
```

Code	Affects	Effect

`body, p {`
`  margin: 0px;`
`}`
	Paragraphs	Remove space between paragraphs

`table {`
`  border-collapse: collapse;`
`}`
	Tables	With `collapse`, adjacent cells share their borders
		
		TABLE border-collapse default value is `separate`

`td {`
`  vertical-align: top;`
`}`
	Table cells	Sets Vertical Alignment in Cells to `top`
		
		TD vertical-align default value is `middle`

Why are empty paragraphs not shown in the result page?

HTML browsers do not show empty/white space only `<p>` elements. Example:

RTF view

```
Line 1
Line 2

Line 3
```

will look different in the HTML browser

Browser view

```
Line 1
Line 2
Line 3
```

You can set the `ConvertEmptyParagraphs` property to true. The result HTML then will contain `<br>` or `<br />` instead of empty `<p>` elements, and look as expected.

Why does the W3C HTML validator show a warning?

The W3C Markup Validation Service[10] displays this warning for a HTML5 document created with ScroogeXHTML using its default settings:

> "Consider adding a `lang` attribute to the `html` start tag to declare the language of this document."

To resolve this warning, enable language conversion and set the default language.

Code example

```
// enable language conversion and set the default language
scrooge.setConvertLanguage(true);
scrooge.setDefaultLanguage("es"); // spanish
```

My HTML document looks different than in the online demo, why?

Short answer: ensure the document is enclosed in outer HTML.

10 https://validator.w3.org/

Long answer: if you set a CSS style sheet using `setStyleSheetInclude`, you must also enable generation of the surrounding HTML code, including the HEAD element. The StyleSheetInclude property is only effective if the property AddOuterHTML is set to true. If the converter property AddOuterHTML is false (the default), the conversion will generate the HTML without the HEAD element.

Code example

```
<p>
  This is a HTML fragment, it is not embedded in outer HTML.
</p>
```

This means it will not contain the CSS style sheet, so the browser will apply its own default style settings.

There are two solutions to fix this:

1. set the property AddOuterHTML to true to ensure your CSS is included in the HEAD element
2. set the CSS within the enclosing HTML document which contains the converted HTML fragment

See Usage of AddOuterHTML

Index

Reference