SCROOGEXHTML
RTF CONVERTER LIBRARIES

Getting started with

ScroogeXHTML RTF Converter

Version 11.5

Copyright (c) Michael Justin

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.

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.

Copyright (c) Michael Justin

Contents

Installation 6
- **Requirements** 6
- **Installation** 6
- **Directory structure** 7
  - Source Edition 7
  - Jar Edition 7
- **Maven and Gradle** 8
  - Maven 8
  - Gradle 8
- **Redistribution** 8

Examples 9
- **"Hello World!"** 9
  - JUnit test 10
- **HTML fragments** 10

Embedding HTML 12
- **Usage of AddOuterHTML** 12
  - Methods converting to files require AddOuterHTML = true 12

Output formats 14
- **Supported output formats** 14
  - HTML5 14
  - XHTML 1.0 Transitional 14
- **Important notes** 15
  - HTML5 documents must use UTF-8 encoding 15

Size optimization 16
- **Default Font Properties** 16
  - Example 16

Font name substitution 17
- **Default font replacing** 17
  - DefaultFontReplacer mappings 17
- **Custom Font Replacing** 17

Table conversion 19
- **Overview** 19

Differences between RTF and HTML tables....19
Vertical cell alignment....19

Hyperlinks....21
Examples....21
Detect hyperlinks based on text attributes....21
Open a hyperlink in a new tab....21
Fix missing https:// in hyperlinks....22

Picture Data Extraction....25
PictureAdapter interface....25
Interface methods....25
Example code....26

Language Support....27
Set the lang Attribute....27
Why you should use the lang attribute on the HTML element....27

Post Processing....28
Overview: Manipulation of the result DOM tree....28
Technical background....28
PostProcessListeners property....28
Performance....28
Example....29
No post processing by default....29
The PostProcessListener interface....29
Examples....30
Add Elements to the HTML head section....30
Complimentary post processors....30

Picture Conversion....32
Complimentary conversion classes....32
Support notice....32
MemoryPictureAdapterDataURI....32
Note: RTF may contain binary picture data....33

Security....34

Experimental Conversion Options....35
HeaderFooterConfiguration#ConvertHeaderAndFooter....35
ListConversionConfiguration#SupportStarPn....35

CSS notes.....36
Overview.....36
Code example.....36

Index.....37

Installation

Requirements

ScroogeXHTML RTF Converter minimum requirements:

- JDK 11+ for development
- Java SE 11+ 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/en/java/javase/11/docs/specs/jar/jar.html
2 http://izpack.org/

Directory structure

Source Edition

```
<inst>
\- addons                                    Add-ons
   \- ...
\- apidocs                                   JavaDoc documentation
   \- ...
\- bom                                       CycloneDX SBOM
\- docs
   \- ScroogeXHTMLGettingStarted.pdf         This document
   \- SECURITY.md                            Security notes
\- examples                                  Examples
   \- ...
\- src
  \- main                                    Library source code
  \- test                                    Test source code
\- pictures                                  Picture support code
   \- ...
\- Uninstaller
   \- uninstaller.jar
.installationinformation
license.txt                                  License information
pom.xml                                      Maven project file
ScroogeXHTML-11.5.0.jar                      Precompiled library
ScroogeXHTML-11.5.0-javadoc.jar              Compressed JavaDoc
ScroogeXHTML-11.5.0-sources.jar              Compressed source code
```

Jar Edition

```
<inst>
\- addons                                    Add-ons
   \- ...
\- apidocs                                   JavaDoc documentation
   \- ...
\- bom                                       CycloneDX SBOM
\- docs
   \- ScroogeXHTMLGettingStarted.pdf         This document
   \- SECURITY.md                            Security notes
\- examples                                  Examples
   \- ...
\- pictures                                  Picture support code
   \- ...
\- Uninstaller
   \- uninstaller.jar
.installationinformation
license.txt                                  License information
ScroogeXHTML-11.5.0.jar                      Precompiled library
ScroogeXHTML-11.5.0-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>11.5.0</version>
</dependency>
```

Gradle

```
implementation 'com.scroogexhtml:ScroogeXHTML:11.5.0'
```

Redistribution

The library (ScroogeXHTML-11.5.0.jar) must not be included with your software as a separate file. For redistribution, integrate it within your application Jar to avoid unauthorized redistribution.

Examples

"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;

public class Tutorial1 {

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

`Output`

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="UTF-8">
    <title>Untitled document</title>
    <meta content="ScroogeXHTML RTF converter 11.5.0" name="generator">
  </head>
  <body>
    <p>"Hello world!"</p>
  </body>
</html>
```

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

JUnit test

This test code verifies that the conversion emits the expected HTML:

Java code

```
    /**
     * Tests the HelloWorld example.
     */
    @Test
    public void testHelloWorld() {
        scrooge.setAddOuterHTML(true);
        String html = scrooge.convert("{\\rtf1 \"Hello world!\"}");
        String expected = String.format(
                "<!DOCTYPE html>\n" +
                "<html lang=\"\">\n" +
                "  <head>\n" +
                "    <meta charset=\"UTF-8\">\n" +
                "    <title>Untitled document</title>\n" +
                "    <meta content=\"ScroogeXHTML RTF converter %s\"
name=\"generator\">\n" +
                "  </head>\n" +
                "  <body>\n" +
                "    <p>\"Hello world!\"</p>\n" +
                "  </body>\n" +
                "</html>\n",
                ScroogeXHTML.getVersion())
                .replace("\n", System.lineSeparator());
        assertEquals(expected, html);
    }
```

Note: the line separator is platform-specific.

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 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 one the methods

- `String convert(String)`
- `String convert(String, Charset)`
- `String convert(ByteArrayInputStream)`
- `String convert(ByteArrayInputStream, Charset)`
- `String convert(Path)`
- `String convert(Path, Charset)`

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.

Methods converting to files require AddOuterHTML = true

If you convert RTF using the methods which take an output path, the AddOuterHTML property must be set to true prior to conversion. Otherwise an exception is thrown.

These methods convert to a file:

- `void convert(String, Path)`
- `void convert(String, Path, Charset)`
- `void convert(Path, Path)`
- `void convert(Path, Path, Charset)`
- `void convert(ByteArrayInputStream, Path)`

Output formats

Supported output formats

HTML5

Document type declaration:

```
<!DOCTYPE html>
```

HTML5 is the default format.

XHTML 1.0 Transitional

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Code example

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

Important notes

HTML5 documents must use UTF-8 encoding

For backwards compatibility, the conversion methods allow to specify an encoding for the conversion.

Please note that HTML5 documents must use UTF-8.

For any other encoding, the w3.org HTML validator on https://validator.w3.org will emit an error message:

> `Error` Legacy encoding windows-1252 used. Documents must use UTF-8.

The conversion fails with a ConversionException if the output document type is set to HTML5 and the conversion method specifies an encoding other than UTF-8.

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 (DefaultFontName as it is after font name substitution, Arial,Helvetica,sans-serif), and set `IncludeDefaultFontStyle` to `true`.

With `AddOuterHTML` set to true, the HTML head section contains this CSS definition:

Code example

```
<style>
  /* generated styles */
  body {font-family:Arial,Helvetica,sans-serif;font-
size:1.17em;color:#000000;}
</style>
```

In this example, whenever a specific part of the text in the document uses a different font size, name, or color than "Arial, 14 pt, black", a span style element will be used to set the correct format. DefaultFontSize is in half-points; the default size unit is em.

Font name substitution

Default font replacing

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

This default implementation uses a list of font names which will be replaced. Replacing is done whenever the font of the text part in the RTF document starts with a name in this list.

DefaultFontReplacer mappings

`Java code`

```
    private Map<String, String> getDefaultReplaceFonts() {
        final Map<String, String> result = new HashMap<>();
        result.put("Arial", "Arial,Helvetica,sans-serif");
        result.put("Courier", "Courier,monospace");
        result.put("Symbol", "Symbol");
        result.put("Times", "Times,serif");
        return result;
    }
```

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

```
@Test
public void testSetFontReplacing() {
    String RTF = "{\\rtf1\\ansi\\ansicpg1025\\deff0\\deflang1033{\
\fonttbl"
            + "{\\f0\\fswiss\\fprq2\\fcharset238 Arial;}}\r\n"
            + "\\f0abc}";
    ScroogeXHTML converter = new ScroogeXHTML();
    converter.setFontReplacing((name, family) -> "Comic Sans");
    String result =
converter.convert(RTF).trim().replace(System.lineSeparator(), "");

    assertEquals("<p>    <span style=\"font-family:Comic
Sans;\">abc</span>  </p>", result);
}
```

Table conversion

Overview

The converter supports conversion of **simple** RTF tables to HTML.

Since version 11.5, the library converts tables by default. Before 11.5, tables in the RTF input document were converted to text paragraphs.

To convert tables to text paragraphs as before, set the ConvertTables property to false. Nested tables are converted only if the ConvertNestedTables property of TablePropConfiguration is enabled (off by default); otherwise the text of a nested table goes into the enclosing cell.

Code example

```
converter.setConvertTables(false);
```

Notes	As some RTF document writers create highly complex RTF code for tables, conversion results may not be perfect.

Differences between RTF and HTML tables

Vertical cell alignment

In HTML, the default value for vertical alignment of table cell content is `MIDDLE`.[3]

The converter assumes that RTF table cell content should be aligned with the top of the cell by default. Cell styles, including the vertical alignment, are written if the

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

ConvertCellStyles option is enabled, the default since version 11.5; top-aligned cells get vertical-align:top.

With ConvertCellStyles off, apply the vertical alignment as the default for table cells with the CSS code `td {vertical-align: top}`.

Code example

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

Hyperlinks

Examples

Detect hyperlinks based on text attributes

Some older RTF documents simply use blue underlined text to indicate hyperlinks. These hyperlinks are not supported by the library. The add-ons include a complimentary (unsupported) post processor in class ConvertUnderlinedToHyperlinks, which may be used as a starting point for your own implementation.

See Post Processing on page 28.

Open a hyperlink in a new tab

With a post process listener, all hyperlinks may be modified to include a target="_blank" attribute.[4]

The following example adds a target="_blank" attribute to all <a> elements which do not have a target attribute.

4 https://stackoverflow.com/q/17711146/80901

Code example

```
scrooge.addPostProcessListener((PostProcessListener) (eventObject) -> {
        try {
            XPathFactory xpathFactory = XPathFactory.newInstance();
            // XPath to find hyperlink nodes.
            XPathExpression xpathExp = xpathFactory.newXPath().compile(
                    "//a[not(@target)]");
            NodeList links = (NodeList)
xpathExp.evaluate(eventObject.getDocument(),
                    XPathConstants.NODESET);
            for (int i = 0; i < links.getLength(); i++) {
                var a = (Element) links.item(i);
                a.setAttribute("target", "_blank");
            }
        } catch (XPathExpressionException ex) {
            ex.printStackTrace(System.err);
        }
    });
```

See also: Post Processing on page 28.

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.

Code example

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

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

        // add post process listener
        scrooge.addPostProcessListener((PostProcessListener) eventObject
-> {
            try {
                XPathFactory xpathFactory = XPathFactory.newInstance();
                // XPath to find hyperlink nodes.
                XPathExpression xpathExp =
xpathFactory.newXPath().compile(
                        "//a[not(contains(@href, '://'))]");
                NodeList links = (NodeList)
xpathExp.evaluate(eventObject.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) {
                ex.printStackTrace(System.err);
            }
        });

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

Output:

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="UTF-8">
    <title>Untitled document</title>
    <meta content="ScroogeXHTML RTF converter 11.5.0" name="generator">
  </head>
  <body>
    <p>
      <a href="https://example.com">example.com</a>
    </p>
  </body>
</html>
```

Picture Data Extraction

PictureAdapter interface

The PictureAdapter interface is designed to support the conversion of embedded pictures within RTF (Rich Text Format) documents. It provides methods for initializing, adding, finalizing, and retrieving pictures during the conversion process. Implementing classes can use these methods to manage the pictures encountered in the RTF code and potentially transform or store them as needed.

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
...
```

Interface methods

Method beforeConvert()

This method is invoked before the conversion of embedded pictures in RTF documents. Implementing classes can override this method to perform any necessary setup or initialization prior to the picture conversion process.

Method addPicture()

Processes and adds an embedded picture encountered during the conversion of RTF documents into a more usable format. Implementing classes should provide the logic to handle the picture data that is being added.

Method getCurrentPicture()

Returns: the current EmbeddedPicture being processed or managed by the implementation.

Method finalizePicture()

Completes any remaining tasks after the processing of an embedded picture during the RTF document conversion process. Implementing classes should use this method to perform any cleanup, finalization, or post-processing needed once an embedded picture has been added, and any necessary transformations have been applied.

Method getPictures()

Returns a map where the keys are unique string identifiers for each picture, and the values are the corresponding EmbeddedPicture objects associated with those identifiers.

Example code

See Picture Conversion on page 32.

Language Support

Set the lang Attribute

The example shown below sets the default language of the document to “en” (English). With AddOuterHTML set to true, the `lang` attribute on the HTML element has the value “en”. ConvertLanguage adds lang attributes to text in other languages.

Code example

```
scrooge = new ScroogeXHTML();
scrooge.setAddOuterHTML(true);
scrooge.setDefaultLanguage("en");
scrooge.getCharPropConvConfig().setConvertLanguage(true);
```

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.getHtmlHeadConfig().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 MyPostProcessListener();

scrooge.addPostProcessListener(listener);

// run the conversion
...
```

No post processing by default

The converter creates and adds no post process handlers by default.

The PostProcessListener interface

The converter stores a list of post process listeners in its `PostProcessListeners` 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.

Examples

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

        scrooge.addPostProcessListener((PostProcessListener) e -> {
            Document doc = e.getDocument();
            Element html = doc.getDocumentElement();
            Node head = html.getFirstChild();
            Element metaDate = doc.createElement("meta");
            metaDate.setAttribute("name", "date");
            metaDate.setAttribute("content", LocalDate.now().toString());
            head.appendChild(metaDate);
        });
        String html = scrooge.convert("{\\rtf1 Hello, World!}");
        System.out.println(html);
    }
```

Output:

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="UTF-8">
    <title>Untitled document</title>
    <meta content="ScroogeXHTML RTF converter 11.5.0" name="generator">
    <meta content="2026-10-19" name="date">
  </head>
  <body>
    <p>Hello, World!</p>
  </body>
</html>
```

Complimentary post processors

MergeBorderDivNodes for paragraph borders

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());
```

Picture Conversion

Complimentary conversion 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.

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.

Note: RTF may contain binary picture data

RTF files which contain picture data may use binary data, and thus there is a risk of data corruption, if the conversion uses one of the conversion methods which take a String as the RTF input.

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/

Security

For security-related configuration please refer to the SECURITY.md document in the docs folder. It describes the conversion guarantees for untrusted RTF, the resource limits (since version 11.4 also the output size and element count), and the lenient and strict modes for malformed RTF.

Experimental Conversion Options

HeaderFooterConfiguration#ConvertHeaderAndFooter

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

By default header and footer will not appear in the output document.

Note	This flag is experimental and unsupported. It is deprecated and will be removed in version 12.0.

ListConversionConfiguration#SupportStarPn

This flag enables support for list formatting based on `\*\pn` RTF tokens (Word 6.0/95 RTF).

Note	This flag is experimental and unsupported.

CSS notes

Overview

Code	Affects	Effect
`body, p {margin: 0;}`	Paragraphs	Remove space between paragraphs. BODY default margin is typically 8px.
`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.`

Code example

Code example

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

Index

Reference

AddOuterHTML....9f., 12, 16
AddPicture....25
BeforeConvert....25
Border-collapse....36
ConvertCellStyles....19
ConvertHeaderAndFooter....35
ConvertNestedTables....19
ConvertPictures....25
ConvertTables....19
CSS....27
Data URI....32
Default Font Properties....16
DefaultFontColor....16
DefaultFontName....16
DefaultFontSize....16
DOM tree....28
Font name substitution....17
FontReplacing....17
Gradle....8
Head....30
HeaderFooterConfiguration....35
HTML5....14, 28
Http....22f.
Https....22
Hyperlinks....21f.
Hyphens:auto....27
IncludeDefaultFontStyle....16
Installation....6
Language Support....27
Margin....36
Maven....8
MergeBorderDivNodes....31
PictureAdapter....25
Post Processing....28
PostProcessEventObject....29
PostProcessListener....23, 28f.
Requirements....6
Size optimization....16
SLF4J....6
SupportStarPn....35
Table....19
TablePropConfiguration....19
Uninstaller....7
UTF-8....15
Vertical-align....36
XHTML....28
XHTML 1.0 Transitional....14
XPath....23