SCROOGEXHTML
RTF CONVERTER LIBRARIES

Getting started with

ScroogeXHTML for the Java™ platform

Version 7.1

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

Page intentionally left blank

Introduction

Features

ScroogeXHTML for the Java™ platform 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)

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 JavaDoc API documentation is located in the installation folder /apidocs.

It is also contained in the ScroogeXHTML-7.1.0-javadoc.jar.

Installation

Requirements

ScroogeXHTML for the Java™ platform requires

- Java SE 7 or 8[1]
- SLF4J (logging framework)
- JDK 7 for development

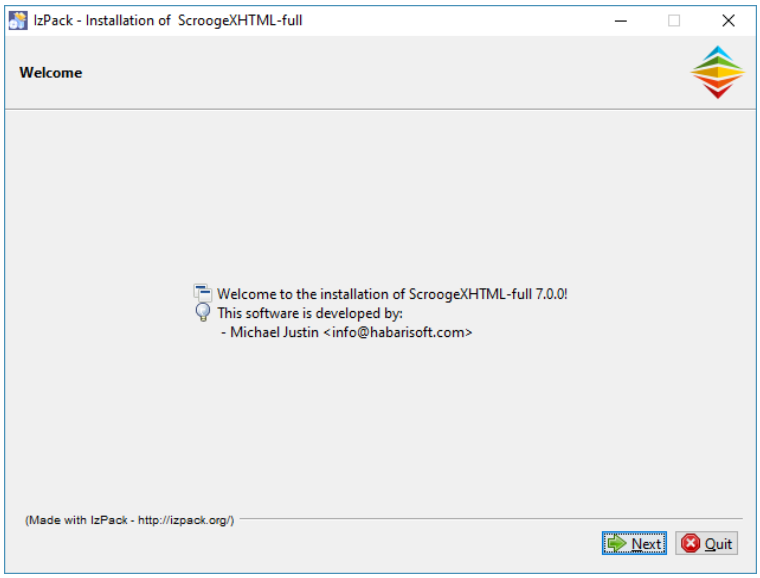
Installation steps

The library installer is an executable JAR[2] file created with izPack[3] and works on Microsoft Windows™, Linux™, Solaris™ and Mac OS X™. A Java Run-time Environment is required to execute it.

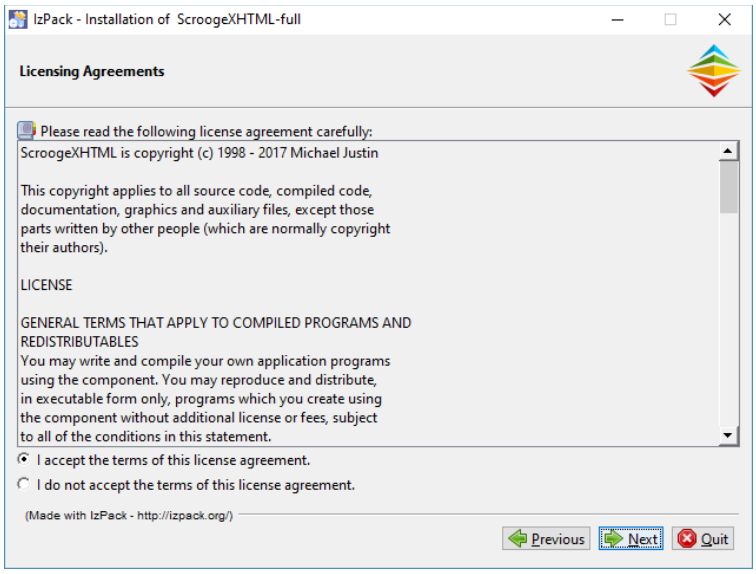
To launch the installer, double-click it. The installer will guide you through the installation steps.

The installation begins with a language selection dialog.

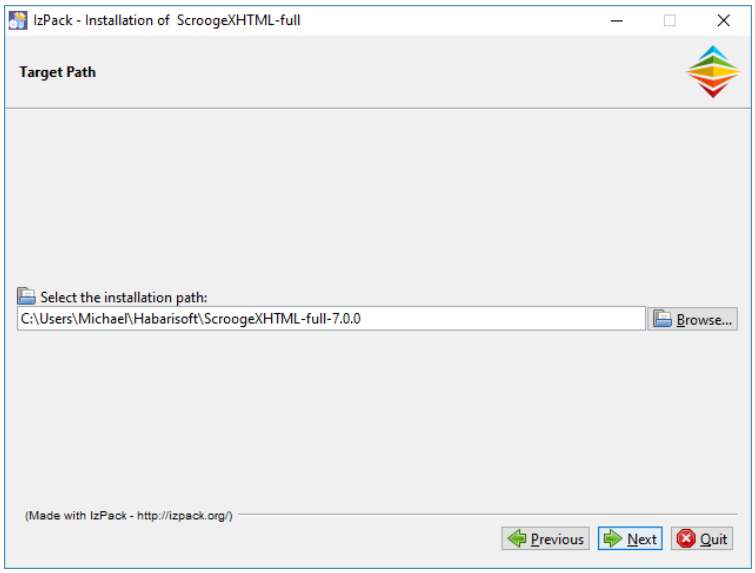
1 The library has not been tested with Java 9
2 https://docs.oracle.com/javase/7/docs/technotes/guides/jar/jarGuide.html
3 http://izpack.org/



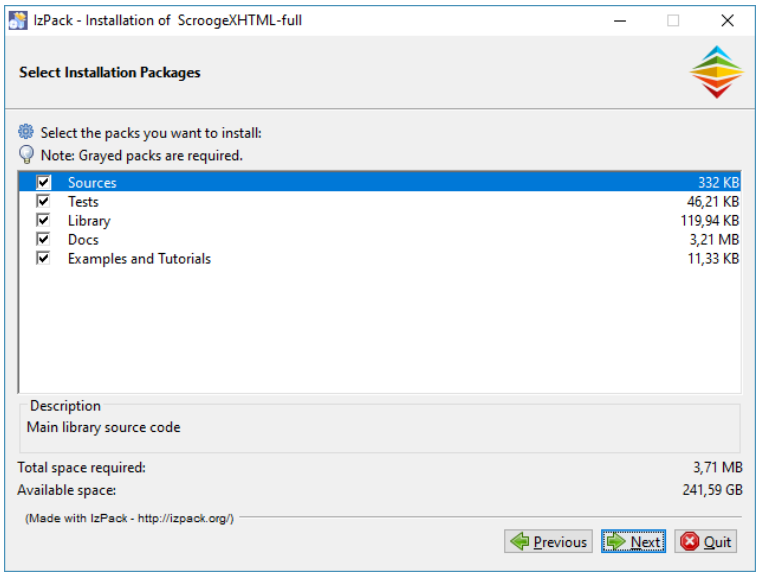
Welcome Page



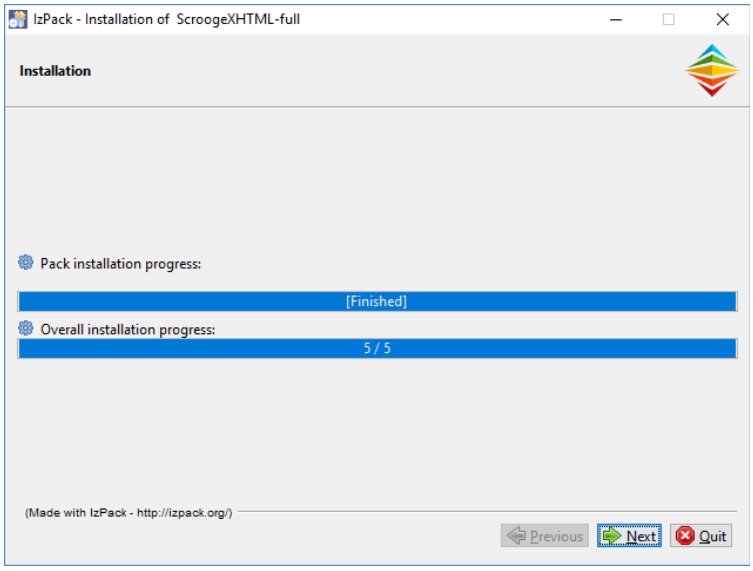
Licensing Agreements



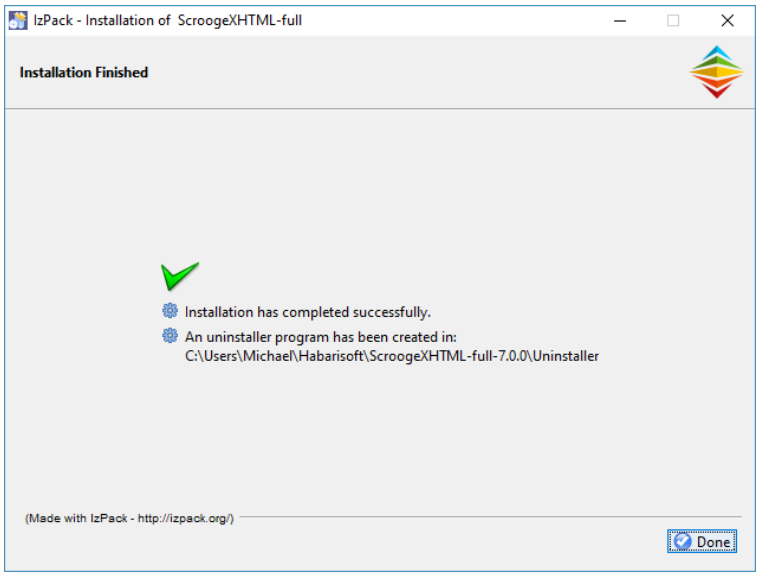
Target Path



Select Installation Packages



Installation



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>7.1.0</version>
  </dependency>
  ...
</dependencies>
```

See also "IDE integration in Maven projects"

Gradle dependency

Gradle

```
dependencies {
    ...
    implementation 'com.habarisoft:ScroogeXHTML:7.1.0'
    ...
}
```

Directory structure

```
<inst>

\- apidocs                                    JavaDoc documentation
   \- ...
\- docs
   \- ScroogeXHTMLGettingStarted.pdf          This document
\- src
  \- main
     \- java
        \- com
           \- habarisoft
              \- scroogexhtml                 Library source code
  \- test
     \- java
        \- com
           \- habarisoft
              \- scroogexhtml                 Test source code
           \- scroogexhtml
              \- example                      Example code
              \- tutorial                     Tutorial code
\- Uninstaller
   \- uninstaller.jar
license.txt
ScroogeXHTML-7.1.0.jar                     Precompiled library
ScroogeXHTML-7.1.0-javadoc.jar             Compressed JavaDoc
ScroogeXHTML-7.1.0-sources.jar             Compressed source code
```

New in version 7.1

New code page mappings

The RTF code page mapping includes new entries:

- Code page 77 mapped to "MacRoman"
- Code page 78 mapped to "SJIS"
- Code page 79 mapped to "CP950"
- Code page 80 mapped to "MS936"
- Code page 179 mapped to "Cp1256"

Symbol font conversion

The library now converts all printable characters of the Symbol font. As a side effect, conversion of bullet lists now correctly creates “•” entity for the leading bullet instead of a “·”.

Support for RTF token _

The library converts the RTF token “_” to a non-breaking hyphen (Unicode value \u2011).

Support for RTF \bullet token

The library converts the \bullet token to a • HTML entity or its Unicode value \u2022.

Breaking changes in version 7.0

ConvertFootnotes default value

The default value of the `ConvertFootnotes` property is **false** now to increase the conversion speed.

Important migration note: if you need footnote conversion, set `ConvertFootnotes` to **true** as shown below:

Code example

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

// footnote conversion is switched off by default in ScroogeXHTML 7.0
scrooge.setConvertFootnotes(true);

// run the conversion
...
```

ConvertHyperlinksForBlueUnderlinedText

The property has been removed as most RTF writers emit hyperlinks encoded as **RTF HYPERLINK** fields.

Hyperlink post processor

The detection and conversion of hyperlinks based on text attributes is still possible with a post processor. An example solution for the conversion of blue and underlined text to hyperlinks is implemented as a post processor (ConvertUnderlinedToHyperlinks).[4]

4 The ConvertUnderlinedToHyperlinks class is provided as unsupported example code

Code example

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

// add the post processor
PostProcessListener l = new ConvertUnderlinedToHyperlinks("#0000ff");
scrooge.getPostProcessListeners().add(l);

// run the conversion
...
```

MetaDateAuto removed

The property was deprecated and has been removed in this version.

No default post process listeners

The converter now does not add any post process listeners by default (on creation) to increase the conversion speed.

The execution of post process listeners may take significant time. Adding post process listeners explicitly in client code (when needed) is more transparent than adding default listeners internally in the converter.

Migration note: if you upgrade from ScroogeXHTML 6.3.0 or later to 7.0, you may add the default progress listeners with this code:

Code example

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

// no post processor are registered by default in ScroogeXHTML 7.0

// add default post processors
scrooge.addDefaultListeners();

// run the conversion
...
```

Progress listener removed

Progress listener methods and properties have been removed to speed up the conversion and to keep the converter library size small.

UseListTable default value

The default value of `UseListTable` is **false** now, since tests showed that many RTF writers use the list table in incorrect or inconsistent ways, which caused low quality conversion results.

In the new default configuration (list table support switched off), conversion results visually match the original document better.

For the reasons described above, the UseListTable property also has been **deprecated** (see below). The new way to enable list table support is shown below:

Code example

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

// list table support is switched off by default in ScroogeXHTML 7.0

// UseListTable is deprecated since 7.0
// scrooge.setUseListTable(true);

// enable RTF list table support when needed:
scrooge.setOutputProperty(ConversionKeys.SUPPORT_LIST_TABLE, "yes");
```

Migration note: the document size increases when list table support is turned off.

Migration from earlier versions

If you migrate from version 6.6 or newer to 7.0, and wish to initialize the converter with compatible default property values, you may use this code:

```
ScroogeXHTML scrooge = new ScroogeXHTML();

scrooge.setOutputProperty(CONVERT_PARAGRAPH_BORDERS, "yes");
scrooge.setOutputProperty(SUPPORT_STAR_PN, "yes");
scrooge.setOutputProperty(SUPPORT_LIST_TABLE, "yes");
scrooge.addDefaultListeners();
scrooge.setConvertFootnotes(true);
```

New in version 7.0

List conversion

Activation of list table support

Starting with version 7.0, the property UseListTable is deprecated. To enable list table support[5], set the OutputProperty SUPPORT_LIST_TABLE to “yes”.

Code example

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

// enable RTF list table support
scrooge.setOutputProperty(ConversionKeys.SUPPORT_LIST_TABLE, "yes");

// run the conversion
...
```

Multi-level list support

Multi-level bullet list conversion support is now available when list table support is enabled.

5 List table support is an unsupported experimental feature

Code example

```
// create a converter instance
ScroogeXHTML scrooge = new ScroogeXHTML();
// enable experimental list table support
scrooge.setOutputProperty(ConversionKeys.SUPPORT_LIST_TABLE, "yes");
// enable experimental multi-level support
scrooge.setOutputProperty(ConversionKeys.SUPPORT_MULTILEVEL, "yes");
// run the conversion
...
```

Numbered lists with roman numbers

Numbered lists with roman numbers are now supported when list table support is enabled.

Experimental feature notice

Important:

- List table support and multi-level list support are experimental features
- If you use them, be aware that not all RTF writers generate correct and consistent list code
- Experimental features may be removed or significantly changed in future releases

Wingdings bullets

The library includes an example post processor which replaces Wingdings bullets with web-safe bullets.

The listener has been designed for usage with RTF generated by WPTools. Other RTF writers might need additional configuration and/or modifications of the listener.

The listener is included with source code[6].

6 The ReplaceWingdingsBullets class is provided as unsupported example code

Code example

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

// non-web-safe bullet used by WPTools 7.2
String badBullet = "Ÿ";

// web-safe bullet
char bullet = '\u2022';

// add the post processor
PostProcessListener l = new ReplaceWingdingsBullets(bullet, badBullet);
scrooge.getPostProcessListeners().add(l);

// run the conversion
...
```

Table conversion

- table border (whole table border) detection improved
- (since 6.6) table cell background color

Other improvements

Paragraph alignment conversion

New property ConvertAlignment (default true)

Paragraph border conversion

Paragraph border conversion is enabled by default. To switch it off, use CONVERT_PARAGRAPH_BORDERS.

Code example

```
scrooge.setOutputProperty(ConversionKeys.CONVERT_PARAGRAPH_BORDERS, "no");
```

Improved support for Japanese text

The range of supported symbols for Japanese text is extended.

JDK 8 Javadoc

Javadoc has been cleaned up to be compatible with the new JDK 8 Doclet.

JavaBean manifest entry

The manifest now includes the JavaBean indicator (JavaBean: true).

Performance improvements

- faster ConvertEmptyParagraphs method
- (since 6.7) faster initialization of DOM tree transformation
- (since 6.6) faster RGB value to HTML color conversion
- (since 6.6) faster cell merging algorithm

Conversion options

Overview

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

Key	Values	Area
SUPPORT_LIST_TABLE[7]	yes | **no**	Lists
SUPPORT_STAR_PN[8]	yes | **no**	Lists
SUPPORT_MULTILEVEL[9]	yes | **no**	Lists
CONVERT_HEADERS_AND_FOOTERS[10]	yes | **no**	Text
CONVERT_PARAGRAPH_BORDERS[11]	yes | **no**	Paragraphs

SUPPORT_LIST_TABLE

This conversion options enables support for the RTF list table (Word 97). 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.

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

7 Experimental, introduced in version 5.3 (UseListTable property)
8 Experimental, introduced in 7.0
9 Experimental, introduced in 7.0
10 Since 6.3.0
11 Since 7.0 it is possible to disable paragraph border box conversion (introduced in version 6.5)

SUPPORT_STAR_PN

This conversion options enables support for list formatting based on *\pn RTF tokens (Word 6.0/95 RTF). This option is experimental. Use with caution.

SUPPORT_MULTILEVEL

This conversion options enables support for multilevel numbered and unnumbered lists in conjunction with SUPPORT_LIST_TABLE (Word 97).

This option is experimental 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.SUPPORT_LIST_TABLE, "yes");

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

// run the conversion
```

CONVERT_HEADERS_AND_FOOTERS

This conversion options enables support for header and footer text conversion. By default header and footer will not appear in the output document.

This option is experimental. Use with caution.

CONVERT_PARAGRAPH_BORDERS

This conversion options enables support for paragraphs which are formatted with a simple border box.

This option is experimental. Use with caution.

Tutorial 1: 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 7.1.0"
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>
```

Source code

The full source code is included in the `com.scroogexhtml.tutorial` package of the library unit tests.

Tutorial 2: 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>
```

Source code

The full source code is included in the `com.scroogexhtml.tutorial` package of the library unit tests.

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.

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`.

Character encoding and document type

Choosing the correct charset[12] and document type (HTML5 or XHTML) for the result document is also important.

Always 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.

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

Hypertext support

Overview

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.

Table conversion

Overview

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);

// run the conversion
...
```

Example for a Data URI link:

Code example

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

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 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.

How to: 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 HowTo3 {

    public static final void main(String[] args) throws IOException {
        String rtf =
"{\\rtf1\\ansi\\ansicpg1252\\deff0\\deflang1031{\\fonttbl{\\f0\\fnil\\fcha
rset0 Tahoma;}{\\f1\\fnil Tahoma;}}\n"
                + "{\\colortbl ;\\red0\\green0\\blue255;}\n"
                + "\\uc1 \n"
                + "\\pard\\cf1\\ul\\f0\\fs16 example.com\\cf0\\ulnone\\f1"
                + "\\par\n"
                + "}";

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

        // add post process listener
        scrooge.getPostProcessListeners().add(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);
            }
        });

        // convert RTF to HTML
        scrooge.setAddOuterHTML(true);
        scrooge.convert(rtf, new File("howto-3.html"));
    }
}
```

How to: fix missing http:// in hyperlinks

Tutorial 3 (included in the unit test folder) 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 is included in the `com.scroogexhtml.tutorial` package of the library unit tests.

Code example

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

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

        scrooge.addPostProcessListener(new StripAttributeLessSpanNodes());

        // 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
        scrooge.setAddOuterHTML(true);
        scrooge.convert(rtf, new File("tutorial-3.html"));
```

Frequently Asked Questions

Conversion

Why are empty paragraphs not in the result page?

HTML browsers do not show empty/whitespace 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.

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

RTF view

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

will look different in the converted HTML document

Browser view

```
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; }
```

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

Index

Reference