



Getting started with

# **ScroogeXHTML for the Java™ platform**

Version 8.1

## **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.

## Contents

<b>Installation.....</b>	<b>5</b>
<b>Requirements.....</b>	<b>5</b>
Android requirements.....	5
<b>Installation.....</b>	<b>5</b>
<b>Directory structure.....</b>	<b>6</b>
<b>Maven and Gradle coordinates.....</b>	<b>6</b>
<b>Tutorial 1: simple conversion.....</b>	<b>7</b>
<b>What it does.....</b>	<b>7</b>
Java code.....	7
Result HTML.....	8
Source code.....	8
<b>Tutorial 2: additional HTML code.....</b>	<b>9</b>
<b>What it does.....</b>	<b>9</b>
Java code.....	10
Result HTML.....	11
Source code.....	11
<b>Embedding HTML.....</b>	<b>12</b>
<b>Usage of AddOuterHTML.....</b>	<b>12</b>
Character encoding and document type.....	12
<b>Hypertext support.....</b>	<b>13</b>
<b>Overview.....</b>	<b>13</b>
Hyperlink field detection.....	13
<b>Table conversion.....</b>	<b>14</b>
<b>Overview.....</b>	<b>14</b>
<b>Vertical Alignment in Cells.....</b>	<b>14</b>
<b>Size optimization.....</b>	<b>16</b>
<b>Default Font Properties.....</b>	<b>16</b>
Example.....	16
<b>Picture extraction.....</b>	<b>17</b>
<b>PictureAdapter Interface.....</b>	<b>17</b>
MemoryPictureAdapter.....	17
MemoryPictureAdapterBase64.....	17

<b>Language support</b> .....	<b>19</b>
<b>How to set the lang Attribute</b> .....	<b>19</b>
Why you should use the lang attribute on the <html> element.....	19
<b>Cascading Style Sheets</b> .....	<b>20</b>
<b>Suggested CSS code fragments</b> .....	<b>20</b>
<b>Post Processing</b> .....	<b>21</b>
<b>Overview: manipulation of the result DOM tree</b> .....	<b>21</b>
Technical background.....	21
PostProcessListeners property.....	21
Performance.....	21
Example.....	22
No post processing by default.....	22
<b>The PostProcessListener interface</b> .....	<b>22</b>
<b>How to: add elements to the HTML head section</b> .....	<b>23</b>
<b>How to: fix missing http:// in hyperlinks</b> .....	<b>23</b>
<b>Advanced conversion options</b> .....	<b>26</b>
<b>Overview</b> .....	<b>26</b>
SUPPORT_LIST_TABLE.....	26
SUPPORT_STAR_PN.....	27
SUPPORT_MULTILEVEL.....	27
CONVERT_HEADERS_AND_FOOTERS.....	27
CONVERT_PARAGRAPH_BORDERS.....	27
<b>Frequently Asked Questions</b> .....	<b>29</b>
<b>Conversion</b> .....	<b>29</b>
Why are empty paragraphs not in the result page?.....	29
<b>Experimental feature notice</b> .....	<b>29</b>
<b>Index</b> .....	<b>30</b>

## Installation

### Requirements

---

ScroogeXHTML for the Java™ platform requires

- Java SE 8
- SLF4J (logging framework)
- JDK 8 for development

### Android requirements

- because of the Java 8 requirement, this library only supports Android 7.0 or later

### Installation

---

The library installer is an executable JAR<sup>1</sup> file created with izPack<sup>2</sup> 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

```

<inst>

\-\- apidocs                               JavaDoc documentation
  \-\- ...
\-\- docs
  \-\- ScroogeXHTMLGettingStarted.pdf      This document
\-\- src
  \-\- main
    \-\- java
      \-\- com
        \-\- scroogexhtml                 Library source code
  \-\- test
    \-\- java
      \-\- com
        \-\- scroogexhtml                 Test source code
        \-\- example                       Example code
        \-\- tutorial                       Tutorial code
\-\- Uninstaller
  \-\- uninstaller.jar
license.txt
ScroogeXHTML-8.1.jar                       Precompiled library
ScroogeXHTML-8.1-javadoc.jar               Compressed JavaDoc
ScroogeXHTML-8.1-sources.jar               Compressed source code

```

## Maven and Gradle coordinates

### Maven

```

<dependency>
  <groupId>com.scroogexhtml</groupId>
  <artifactId>ScroogeXHTML</artifactId>
  <version>8.1</version>
</dependency>

```

### Gradle

```
implementation 'com.scroogexhtml:ScroogeXHTML:8.1'
```

## 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 8.1"
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);
        }
    }
}

```

## ScroogeXHTML for the Java™ platform 8.1

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<sup>3</sup> and document type (HTML5 or XHTML) for the result document is also 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
------	--------------------------------------------------------------------------------------------------------------------------------------------------------------

---

<sup>3</sup> <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 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 simple tables, set the `ConvertTables` property to true.

### Code example

```
scrooge.setConvertTables(true);
```

### Note

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

## Vertical Alignment in Cells

---

In HTML, the default value for vertical alignment of table cell content is "middle".<sup>4</sup>

ScroogeXHTML for the Java™ platform 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.

---

<sup>4</sup> See <https://stackoverflow.com/questions/33487148/>

## ScroogeXHTML for the Java™ platform 8.1

### Code example

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

# 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>
```



## 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

```

```

The image URLs will be numbered automatically.

The class allows to set a base path with `setBase(String base)`, for example `adapter.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

```

```

## Language support

### How to set the lang Attribute

---

Example sets the default language to "en" (English). This adds the `lang` attribute on the `html` element to "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).

Add `hyphens: auto` in your CSS:

#### Code example

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

### References

<http://blog.adrianroselli.com/2015/01/on-use-of-lang-attribute.html>

<https://www.w3.org/International/questions/qa-lang-why>

# Cascading Style Sheets

## Suggested CSS code fragments

Suggestions for your custom CSS in documents which use tables:

### 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	Reference
<pre>body, p {   margin: 0px; }</pre>	Reduce space between lines
<pre>table {   border-collapse: collapse; }</pre>	TABLE collapse default is "separate", with collapse, adjacent cells share their borders
<pre>td {   vertical-align: top; }</pre>	<a href="#">Vertical Alignment in Cells</a> (HTML default is "middle")

## 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 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);
    }
}
```

## How to: fix missing http:// in hyperlinks

---

This example 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.



## ScroogeXHTML for the Java™ platform 8.1

### 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(
                "//*[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"));
```

## Advanced 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 <sup>5</sup>	yes   <b>no</b>	Lists
SUPPORT_STAR_PN <sup>6</sup>	yes   <b>no</b>	Lists
SUPPORT_MULTILEVEL <sup>7</sup>	yes   <b>no</b>	Lists
CONVERT_HEADERS_AND_FOOTERS <sup>8</sup>	yes   <b>no</b>	Text
CONVERT_PARAGRAPH_BORDERS <sup>9</sup>	yes   <b>no</b>	Paragraphs

### 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 because some RTF writers generate malformed list tables. Use with caution.

<sup>5</sup> Experimental, introduced in version 5.3 (UseListTable property)

<sup>6</sup> Experimental, introduced in 7.0

<sup>7</sup> Experimental, introduced in 7.0

<sup>8</sup> Since 6.3.0

<sup>9</sup> Since 7.0 it is possible to disable paragraph border box conversion (introduced in version 6.5)

## **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. Use with caution.
------	------------------------------------------------

## **SUPPORT\_MULTILEVEL**

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

Note	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 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".

## **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");
```

If consecutive paragraphs use the same border style, they can be merged with a post processor.<sup>10</sup>

### Code example

```
// merge identical paragraph borders
scrooge.addPostProcessListener(new MergeBorderDivNodes());
```

This post processor removes all identical style (div) nodes between paragraphs and replaces them with one parent div node.

---

<sup>10</sup> the class is included in `<inst>src/test/java/com/scroogexhtml/tidy`

## Frequently Asked Questions

### Conversion

---

#### Why are empty paragraphs not 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.

### 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

# Index

## Reference

AddOuterHTML.....	7, 12, 16	Hyperlinks.....	23
ConvertEmptyParagraphs.....	29	Hypertext.....	13
ConvertFields.....	13	IncludeDefaultFontStyle.....	16
ConvertHyperlinks.....	13	Installation.....	5
ConvertPictures.....	17	MemoryPictureAdapter.....	17
ConvertTables.....	14	MemoryPictureAdapterBase64.....	17
Data URI.....	17	PictureAdapter.....	17
DefaultFontColor.....	16	PostProcess.....	22
DefaultFontName.....	16	PostProcessEventObject.....	22
DefaultFontSize.....	16	PostProcessListener.....	21p.
Fragment.....	9	SLF4J.....	5
Head.....	23	Tutorial.....	7
Hidden fields.....	13	UnsupportedOperationException.....	12
HTML5.....	9, 21	XPath.....	24
Http.....	24	.....	29
Hyperlink.....	13		