JWord 3

User's Guide

V3.4.x / July 2015

Introduction

JWord is an advanced rich text editing component for the Java platform. Typical applications where JWord can be used are WYSIWYG text editing, content authoring, reporting and mail-merge (document merge).

JWord is based on the Java Swing Text API, which is a framework for developing different text editors. Refer to the Swing documentation for the Text API related information.

The basic classes included by the JWord package are:

- *JWordTextPane*: A Swing component that displays the document and controls edit actions
- *JWordDocument*: Keeps and manages the document data (model)
- *JWordController*: Initiates and controls the user interfaces (frames, dialogs etc).
- *JWordEditorKit*: Handles editor functionality that is not dependent on the component. Document initialization, I/O, action set and view generation are among the functionality provided by this class.

The package contains various view, utility, model classes, user interface resources and I/O related classes along with the above ones.

You can directly add *JWordTextPane* to your Swing windows or you can use the fully decorated user-interface with menus, ruler, toolbars as a separate frame or dialog. You can even add the UI completely or partially to your own Frame's, InternalFrame's, Panel's and Dialog's.

This document is a guide to using JWord in your applications. Refer to the javadocs for class documentation.

Requirements

JWord is based on the J2SE platform. JRE1.6 or more is required for the package to work.

Using As A Separate Frame/Dialog

In order to create a separate frame window with the *JWordTextPane* component, a *JWordController* must be constructed and initiated. You can then customize the UI by setting some options such as the state of menus, toolbars, ruler etc.

Opening a JWord Frame:

```
try
{
   JWordController ctrl=new JWordController();
   ctrl.init(false);
   //ctrl.setShowRuler(false);
   //ctrl.setShowToolbar1(false);
   //ctrl.setShowToolbar2(false);
   //ctrl.setShowMenu(false);
   //ctrl.setShowStatusBar(false);
   //ctrl.setShowMenuItem(JWordController.mi_open, false);
   //ctrl.setShowToolbarItem(JWordController.tb_open, false);
   //ctrl.getEditor().setViewMode(JWordTextPane.VM_NORMAL, true);
   //ctrl.setHtml;
   ctrl.getFrame().setVisible(true);
   ctrl.getEditor().requestFocus();
}
catch (GuiException e)
{
   e.printStackTrace();
}
```

Opening a JWord Dialog:

The controller initiates an invisible *JFrame* first. By moving components from frame to the empty dialog, you can have a dialog decorated with the editor interfaces.

```
try
{
   JWordController ctrl=new JWordController();
   ctrl.init(false);
   JDialog dialog=new JDialog();
   dialog.setSize(400, 600);
   dialog.setJMenuBar(ctrl.getFrame().getJMenuBar());
   dialog.setContentPane(ctrl.getFrame().getContentPane());
   dialog.setVisible(true);

   ctrl.getEditor().requestFocus();
}
catch (GuiException e)
{
   e.printStackTrace();
}
```

Opening the Controller in HTML Editing Mode:

The HTML Editing mode is customized for editing XHTML files. In this mode, tabs, section formatting, header/footers, soft-breaks, flow control options and some bullet symbols are not present. Default table and column width is in %.

HTML editing mode is an attribute of the *JWordEditorKit*. This attribute can be set when the controller is initialized.

```
JWordController ctrl=new JWordController(true,true,true,true,null);
```

The controller can also be initialized with an editor kit:

```
JWordEditorKit kit=new JWordEditorKit(ctx);
kit.setHtmlMode(true);
JWordController ctrl=new JWordController(kit, null);
```

Controller Options:

The user interface can be customized by some options which are input to the constructor of the controller:

wmfSupport:	Enable WMF/EMF images.
svgSupport:	Enable SVG images.
equationSupport:	Enable equation (JMathEdit component) support.
htmlMode:	Work in HTML editing mode.
pdfSupport:	Enable exporting as PDF (requires Apache FOP).
spellCheckSupport:	Enable spell checking.
forcedLocale:	Use the given locale for finding resources.

The Controller creates an EditorKit and TextPane but it can also be initialized with a constructed EditorKit. The textpane can be set/changed after the controller initializes using the *setEditor( )* method. This way multiple text panes can be controlled with a single controller.

If SVG images are enabled, the editor kit uses the default renderer for displaying SVG images. If Apache Batik will used for rendering SVG images, *SVG_RENDERER* static value of *JWordEditorKit* must be set to "*com.pilot.jword.view.SvgIcon*" and Batik jars must be included in the application.

Exporting as PDF requires Apache FOP libraries in the classpath. If classpath size is an issue, you may prefer to export as XSL-FO, and then use FOP on the server side to produce the final PDF.

Using Inside Other Frames/Dialogs

Just like other Swing components, *JWordTextPane* can be added to other Swing containers. In order to create a new document, first an instance of *JWordEditorKit* must be created. You can keep a singleton instance of this object in your application.

In most cases you would like to add the editor component into a scrolled pane. The following code snippet creates an editor component inside a *ScrolledPane* and initializes it with an empty document.

```
import java.awt.BorderLayout;
import java.awt.Container;

import javax.swing.JFrame;
import javax.swing.JScrollPane;

import com.pilot.jword.JWordDocument;
```

```
import com.pilot.jword.JWordEditorKit;
import com.pilot.jword.JWordTextPane;

public class EditorFrame extends JFrame
{
   private JWordTextPane textComp;
   public static JWordEditorKit editorKit;

   public static void main(String[] args)
   {
      editorKit=new JWordEditorKit();
      EditorFrame editor = new EditorFrame();
      editor.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      editor.setVisible(true);
   }

   // Create the editor.
   public EditorFrame()
   {
      super();
      textComp = createTextComponent();
      JScrollPane scrollEdit = new JScrollPane(textComp);
      Container content = getContentPane();
      content.add(scrollEdit, BorderLayout.CENTER);
      setSize(320, 540);
   }

   // Create the JWordTextPane.
   protected JWordTextPane createTextComponent()
   {
      JWordDocument doc=(JWordDocument) editorKit.createDefaultDocument();
      JWordTextPane tc=new JWordTextPane(doc);
      tc.setViewMode(JWordTextPane.VM_NORMAL, true);
      tc.setEditorKit(editorKit);
      ((JWordDocument) tc.getDocument()).init();

      return tc;
   }
}
```

Modifying The Document

Once you have the handle of a *JWordTextPane* by any of the methods mentioned above, you can use this handle to modify the attached document from your application. You can insert text into any location in the document, you can add tables, images, symbols, modify selection range, alter properties of the selected range, etc.

Almost all dimensions in JWord, such as font sizes, indenting amounts, table widths etc. are specified in points or twips (20th of a point). Following the architecture of the Swing Text API, attributes are transferred to methods in *AttributeSet*'s. Some data structures are also defined and passed to methods for convenience. JWord specific attributes should be set using the *StyleConstantsExt* class, the rest should be set using the *StyleConstants* class of the framework.

Bulk text insertion:

Use the following method provided by the *JWordTextPane* to insert formatted bulk text, including tables, symbol and other elements. This method allows developer to insert a tree of elements that represent the piece of text in a tree structure. *ContentTree* class has methods to create new elements of different types and attributes under other elements to construct a tree.

Method	Description
insert(int,ContentTree,boolean,int,int)	Insert bulk text into the given location.

Example:

```
JWordTextPane  editor;
//...

//create paragraph attribs
SimpleAttributeSet para1=new SimpleAttributeSet();
StyleConstants.setAlignment(para1, StyleConstants.ALIGN_LEFT);
StyleConstants.setSpaceAbove(para1,500);
StyleConstants.setSpaceBelow(para1,500);

SimpleAttributeSet para=new SimpleAttributeSet();
StyleConstants.setAlignment(para, StyleConstants.ALIGN_CENTER);
StyleConstants.setSpaceAbove(para,0);
StyleConstants.setSpaceBelow(para,0);
//create character attribs
SimpleAttributeSet charA=new SimpleAttributeSet();
StyleConstants.setFontFamily(charA, "Calibri");
StyleConstants.setFontSize(charA, 200);
```

```
StyleConstants.setBold(charA, true);

//create table attribs
SimpleAttributeSet table=new SimpleAttributeSet();
ArrayList<TableColumn> cols=new ArrayList<TableColumn>();
for(int j=0;j<5;j++)
   cols.add(new TableColumn(1000,StyleConstantsExt.UNIT_PHYSICAL));
StyleConstantsExt.setTableColumns(table, cols);
StyleConstantsExt.setTableLeftIndent(table, 0);
StyleConstantsExt.setWidth(table, new Dimension(1000*5,StyleConstantsExt.UNIT_PHYSICAL));
Borders brd=new Borders(20,Color.green);
StyleConstantsExt.setBorders(table, brd);
Insets ins=new Insets(0,0,0,0);
StyleConstantsExt.setCellMargins(table, ins);
//create dummy row/cell attribs
SimpleAttributeSet row=new SimpleAttributeSet();
SimpleAttributeSet cell=new SimpleAttributeSet();

//create a new content tree
ContentTree ct=new ContentTree();
//add a line
ct.startParagraph(para1);
ct.addContent(charA, "List of players");
ct.endParagraph();

//add a table
ct.startTable(table);
for(int i=0;i<30;i++)
{
   ct.startTableRow(row);
   for(int j=0;j<5;j++)
   {
      ct.startTableCell(cell);
      ct.startParagraph(para);
      ct.addContent(charA, "cell["+i+","+j+"]");
      ct.endParagraph();
      ct.endTableCell();
   }
   ct.endTableRow();
}
ct.endTable();
//add last line
ct.startParagraph(para);
ct.addContent(charA, "");
ct.endParagraph();
//insert content
try
{
   editor.insert(0, ct, false, 0, JWordDocument.DEST_BODY);
}
catch(BadLocationException e)
{
   e.printStackTrace();
}
```

Text modifications:

Text can be modified though the *JWordDocument* methods. There are some shortcuts in the *JWordTextPane* for some frequently used functionality.

List of *JWordTextPane* methods that manipulate text in document:

Method	Description
insertString(int,String,AttributeSet)	Inserts a text into a given location, with the specified attributes
removeText(int,int)	Removes text from the document, for the given range
removeSelection()	Removes selected text from document
getCharacterAttributes(int)	Returns an attribute-set containing character attributes of the given offset
findReplace(int,String,String,boolean,boolean)	Searches for occurrences of the given search text in the document, and replaces with the given text
findReplaceAll(int, String, String, boolean, boolean)	Searches for all occurrences of the search text and replaces with the given text
findTextOffset(int, int, String, boolean, boolean)	Returns the offset of the first occurrence of the given search text
getParagraphAttributes(int)	Returns an attribute-set containing paragraph attributes of the given offset
getSelectionCharProps()	Returns mutual character attributes of the selected region.
getSelectionParagraphBorderAttributes()	Returns mutual paragraph border attributes of the selected region.
getSelectionParagraphNumberingAttributes()	Returns mutual paragraph numbering attributes of the selected region.
getSelectionParagraphProps()	Returns mutual paragraph attributes of the selected region.
setCharacterAttributes(AttributeSet, AttributeSet)	Sets character attributes of the selected region. Can also remove some attribs
setCharacterAttributes(int, int, AttributeSet)	Sets character attributes of the given document range
setParagraphAttributes(int, int, AttributeSet)	Sets paragraph attributes of the given document range

setParagraphMargins(int, int, int, int,int)	Sets paragraph margins of the given range
setSelectionCharProps(CharacterProperties)	Sets mutual character attributes of the selected range
setSelectionCharProps(CharacterProperties, CharacterProperties)	Sets mutual character attributes of the selected range, by comparing with the old attributes. Only changed attributes will be modified.
setSelectionParagraphBorderAttributes(BordersAndShadingProperties)	Sets paragraph border attributes of the selected range
setSelectionParagraphNumberingAttributes(NumberingProperties)	Sets paragraph numbering attributes of the selected range
setSelectionParagraphProps(ParagraphProperties)	Sets mutual paragraph attributes of the selected range
setSelectionParagraphProps(ParagraphProperties, ParagraphProperties)	Sets mutual paragraph attributes of the selected range, by comparing with the old properties. Only changed attributes will be modified.
setSelectionStyle(String)	Sets style of the selected region.
setSelectionTabs(TabSet)	Sets tab-set of the selected range
toggleSelectionBulletState()	Toggles bulleting state of the selected paragraphs
toggleSelectionNumberState()	Toggles numbering state of the selected paragraphs
setHyperLink	Adds a hyperlink to the given range
increaseIndent	Increases indent or numbering level of the given range
decreaseIndent	Decreases indent or numbering level of the given range

Here is a code sample for inserting text into some location in the document:

```
JWordTextPane  editor;
//...
try
{
   //create attributes
   SimpleAttributeSet attrs=new SimpleAttributeSet();
   StyleConstants.setFontFamily(attrs,"Calibri");
   StyleConstants.setFontSize(attrs,200);//200=10 pt in twips
   StyleConstants.setItalic(attrs, true);
   //insert text
   editor.insertString(0, "This is a test string\n", attrs);
}
catch(BadLocationException e)
{
   e.printStackTrace();
}
```

Another sample, which changes selected text to bold, and colored to red:

```
JWordTextPane  editor;
//...
//get attributes
CharacterProperties cp=editor.getSelectionCharProps();
cp.isBold=true;
cp.foreGround=Color.red;
cp.fgAuto=false;

//modify selection
editor.setSelectionCharProps(cp);
```

Inserting images, symbols, fields:

JWordTextPane has the following methods that insert images, symbols and fields.

Method	Description
getImageProperties()	Return properties of the selected image
insertDateTime(int,String,boolean,byte)	Inserts a date-time field into the given offset
insertImage(int,Icon,int)	Inserts an image into the given document offset
insertPageNo(int)	Inserts “page-no” field at the given document offset
insertSymbol(int,String,Character)	Inserts a unicode symbol, with given font face, into the given document offset
setImageAttributes(ImageProperties)	Modify properties of the selected image
setImageAttributes(int,AttributeSet)	Modify image attributes at the given document offset
insertEquation	Inserts a JMathEdit equation into the given position

İmage example:

```
try
{
   ImgIcon icon = new ImgIcon("c:\\temp\\test.png");
   editor.insertImage(-1, icon, StyleConstantsExt.VA_BOTTOM);
}
catch (BadLocationException e)
{
   e.printStackTrace();
}
```

Date-time field example:

```
try
{
   editor.insertDateTime(-1,"dd.MM.yyyy",true,FieldDateTime.DATE);
}
catch (BadLocationException e)
{
   e.printStackTrace();
}
```

Table operations:

List of *JWordTextPane* methods for inserting, altering and removing tables is below. Tables can be inserted anywhere in the document. Nesting of tables is also supported.

Method	Description
deleteColumn()	Deletes the current table column, at the caret position
deleteRow()	Deletes the table row at the caret position
deleteTable()	Deletes the table at the caret position
getSelectionCellBorderAttributes()	Returns cell border attributes of the selected cell
getTableProperties()	Returns table attributes of the table at the caret position
insertColumn(boolean)	İnserts a new column into the table at the caret position
insertRow(boolean)	Inserts a new row into the table at the caret position
insertTable(int,int,int,Insets, Borders,int,int,Color)	Inserts a new table into the given position with the given attributes
isCaretInTable	Checks whether the caret is inside a table
mergeCells(int,int,int,int)	Merges the table cells in the range
mergeSelectedCells	Merges selected cells
setSelectionCellBorderAttributes(BorderAndShadingProperties, BorderAndShadingProperties, boolean,boolean,boolean)	Sets cell borders of the cells in the selection, by comparing with the old values.
setTableProperties(TableProperties)	Sets table (table, row, and cell) properties for the selected range.
splitCells(int)	Splits cells in the given offset.

A code snippet that inserts a new table:

```
try
{
  Insets cellMargins=new Insets(0,0,0,0);
  Borders border=new Borders(20,Color.black); //1 pt, black borders on all sides
  editor.insertTable(-1, 3, 3, cellMargins, border, StyleConstantsExt.HA_LEFT, 0, Color.white);
}
catch(BadLocationException e)
{
  e.printStackTrace();
}
```

A code snippet that alters width of second column of the table at the caret position:

```
//get table properties for the caret position
TableProperties tp=editor.getTableProperties();
//get column 2
TableColumn tc=tp.columns.get(1); //zero based
//change width
tc.width.value=1000; //in twips
tc.width.unit=StyleConstantsExt.UNIT_PHYSICAL;
//set table properties
editor.setTableProperties(tp);
```

Hyperlinks:

JWordTextPane allows specifying hypelink attribute for leaf elements such as labels, images, equations and fields. *JWordTextPane* methods for setting hyperlink attributes are:

Method	Description
setHyperLink	Adds a hyperlink to the given range
getHyperLink	Get hyperlink attribute of the given position
addHyperLinkListener	Register listeners for hyperlink actions

In order to determine the action to take when an hyperlink is visited, you can call the *addHyperLinkListener*(*HyperlinkListener* l) method of *JWordTextPane* and provide a listener for hyperlink events. In the editor area when CTRL key is pressed, hyperlinks can be visited by clicking the left button. This action will fire an hyperlink event, and inform the registered listeners.

Section, header-footer, column, page modifications:

JWordTextPane methods for modifying document structure are:

Method	Description
getColumnSettings(int)	Returns page column attributes for the given document offset
getPageSettings(int)	Returns page settings for the given document offset
setPageSettings(int, PageSettings)	Sets page attributes at the given document offset
insertColumnBreak(int)	Inserts a column soft-break at the given offset
insertPageBreak(int)	Inserts a page soft-break at the given offset
insertSection(int, ColumnSettings, boolean)	Starts a new section at the given offset with the specified properties.
insertHeaderFooter()	Inserts header & footer

Loading/Saving/Printing

Document initialization and I/O operations are the responsibility of the *JWordEditorKit* class. Here is the list of related methods in this class:

Method	Description
createDefaultDocument()	Creates a new document
newDocument(JWordTextPane)	Create a new document and attaches to a *JWordTextPane*
read(InputStream,Document,int)	Loads RTF data from the given stream into the given offset of the given document
read(Reader, Document, int)	Loads RTF data from the given Reader into the given offset of the given document
read(Reader, Document ,int, boolean)	Loads RTF data from the given Reader into the given offset of the given document
readXML(InputStream, Document ,int)	Loads XML data from the given stream into the given offset of the given document
readODF(InputStream, Document, int)	Loads ODF data from the given stream into the given offset of the given document
readXHTML(String fname, Document doc, int pos)	Loads XHTML data from the given file into the given document. (XHTML file must be created by JWord, and include XML representation)
write(OutputStream, Document, int, int, boolean)	Writes the specified part of the document into the given stream, in RTF format
write(String, JWordDocument)	Saves the given document to a file in RTF
write(Writer, JWordDocument, int, int, boolean)	Writes the specified part of the document, using the given writer, in RTF
writeODF(String, JWordDocument)	Saves the document to a file in ODF
writeODF(OutputStream, JWordDocument, int, int, boolean)	Writes the specified part of the document, into the given stream, in ODF
writeXML(String, JWordDocument)	Saves the document to a file in XML
writeXML(OutputStream, JWordDocument, int, int, boolean)	Writes the specified part of the document, into the given stream, in XML
writeXHTML(String fileName, JWordDocument doc,int svgOption,int wmfOption,int equationOption,int dpi,int imageDpi,boolean useObjectTagForSvg)	Saves the document as an XHTML file.
writePDF(OutputStream, JWordDocument)	Exports the document as PDF (requires Apache FOP in the classpath)
writePDF(String, JWordDocument)	Exports the document as PDF to the given file (requires Apache FOP in the classpath)
writeXSLFO(Writer, JWordDocument)	Exports the document as XSL-FO using the given writer
writeXSLFO(String, JWordDocument)	Exports the document as XSL-FO to the given file
writeTXT(String, JWordDocument)	Exports the document as plain TXT to the given file
writeTXT(Writer,int,int,boolean)	Exports part of the document using the given writer.
writeTXT(OutputStream,int,int,boolean)	Exports part of the document to the output stream.

JWord's current version supports RTF version 1.9 and ODF version 1.2. For working with SVG images documents must be saved in XML format since neither of the other formats support SVG. The schema for JWord XML output can be found in com/pilot/jword/jword.xsd file.

Example1, Loading an XML file:

```
//create a new document
JWordEditorKit kit=(JWordEditorKit) editor.getEditorKit();
JWordDocument dnew = (JWordDocument) kit.createDefaultDocument();
//open file
FileInputStream in = new FileInputStream("/usr/home/docs/doc1.xml");
//read into document
kit.readXML(in, dnew, 0);
//close stream
in.close();
//set document in editor
editor.setDocument(dnew);
```

Care should be taken when this type of code must be called from a thread other than the main event handling thread. *invokeLater* method of *SwingUtilities* class should be used from other threads:

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
      //actual work
    }
  });
```

Printing a document:

You can directly call *JWordTextPane's print*() method to print the document, by first inputting the printer settings through a dialog. You can also create a *Graphics* instance using your own printer settings and call *print*() with this *Graphics* context without a dialog opening. JWord provides a *getPrintable()* method, which allows printing to a print job created by other means. This is also useful in printing to virtual printers:

```
PrinterJob pJob = PrinterJob.getPrinterJob();
      try
      {
         pJob.setPrintable(editor.getPrintable(null, null));
         if (pJob.printDialog())
            pJob.print();
      }
      catch (PrinterException ex)
      {
         showErrorMessage("print error");
      }
```

Styles

Styles allows users to store AttributeSet's by their names. By attaching document elements to styles, it is possible to change the look of all the elements together. Styles are chained in parent-child relation. JWord defines character and paragraph styles which are applicable to labels, icons, fields and paragraphs.

JWordTextPane provides two methods to create new styles and modify existing ones:

Method	Description
addNamedStyle (String, int, String, String, CharacterProperties, ParagraphProperties, BordersAndShadingProperties, NumberingProperties)	Creates a new style
modifyNamedStyle(String, String, CharacterProperties, ParagraphProperties, BordersAndShadingProperties, NumberingProperties)	Modfies style attributes

It is a good convenience to open the editor with a predefined style library. Like the following code sample below, you can create your own style library programmatically and start the editor with the library. It is also possible to save this document as template document and create new documents by opening the template

```
JWordTextPane editor=ctrl.getEditor();
//redefine default
CharacterProperties cpDefault=new CharacterProperties();
ParagraphProperties ppDefault=new ParagraphProperties();
cpDefault.fontFace="Arial";
cpDefault.fontSize=180; //9 pts
ppDefault.alignment=StyleConstants.ALIGN_LEFT;
editor.modifyNamedStyle("default", "default", "default", cpDefault, ppDefault, null, null);
//define heading1
CharacterProperties cpH1=new CharacterProperties();
ParagraphProperties ppH1=new ParagraphProperties();
BordersAndShadingProperties brdH1=new BordersAndShadingProperties();
cpH1.fontFace="Arial";
cpH1.fontSize=280; //14 pts
ppH1.spaceBefore=140.0f;
ppH1.spaceAfter=140.0f;
brdH1.borders=new Borders(0,Color.black);
brdH1.borders.setBorders(Borders.BOTTOM, 20, Color.black);
editor.addNamedStyle("heading1", StyleConstantsExt.PARA_STYLE, "default",
   "default", cpH1, ppH1, brdH1, null);
//define heading2
CharacterProperties cpH2=new CharacterProperties();
ParagraphProperties ppH2=new ParagraphProperties();
cpH2.fontFace="Arial";
cpH2.fontSize=220; //11 pts
cpH2.isBold=true;
editor.addNamedStyle("heading2", StyleConstantsExt.PARA_STYLE, "default",
   "default", cpH2, ppH2, null, null);
//define emph
```

```
CharacterProperties cpEmph=new CharacterProperties();
cpEmph.isBold=true;
editor.addNamedStyle("emph", StyleConstantsExt.CHAR_STYLE, "default", "default",
   cpEmph, null, null, null);
//define strong
CharacterProperties cpStrong=new CharacterProperties();
cpStrong.isBold=true;
cpStrong.isUnderline=true;
editor.addNamedStyle("strong", StyleConstantsExt.CHAR_STYLE, "default", "default",
    cpStrong, null, null, null);
```

Spell Checking

JWord allows automatic and manual spell checking by plugging-in third party or custom spell checking or dictionary classes. You can create custom spell checkers which can rely on external services, use special dictionaries related to a specific area or combine various dictionaries in one. You can also integrate a third party spell checker with complete dictionaries. JWord will handle highlighting of spelling errors, suggesting words and interacting with the spell checker as the user changes, updates, and ignores spelling errors.

In order to enable spell checking in JWord, first *JWordController* must be initialized with *spellSupport=true* option. An *EditorSpellChecker* instance must be supplied to the *JWordTextPane*. An instance of *SpellChecker* interface will be needed to construct the *EditorSpellChecker*:

```
JWordController ctrl=new JWordController(true,true,true,false,true,true,null);

ctrl.init(false);
EditorSpellChecker sc=new EditorSpellChecker(Locale.ENGLISH, new JMySpellerTestSpeller());
ctrl.getEditor().setSpellChecker(sc);
ctrl.getEditor().autoSpellCheck(true);

ctrl.getFrame().setVisible(true);
ctrl.getFrame().setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
ctrl.getEditor().requestFocus();
```

The *SpellChecker* interface requires the following methods to be implemented:

```
public void init(Locale locale); //initialize checker

public boolean checkWord(String word); //check a word

public String[] suggest(String word); //suggest words for a spelling error

public void addToDictionary(String word); //add word to dictionary
```

Headless Mode

Some functionality in JWord can also be useful in headless mode where the component is not visible. Server side services, batch operations are examples of such usage. It is possible to open or create documents, modify document content programmatically, populate a mail-merge data, then save or print in various formats without needing to showing up the pages.

The JVM can be run in “headless mode” by supplying the *-Djava.awt.headless=true* runtime arguments. In headless mode, *JWordEditorKit, JWordDocument* and View classes are still available. Here is a code sample that loads a document and then exports to PDF:

```
JWordEditorKit kit = new JWordEditorKit();

JWordDocument dnew = (JWordDocument) kit.createDefaultDocument();
try
{
   kit.read("C:\\temp\\doc.rtf", dnew, 0);
   kit.writePDF("C:\\temp\\doc.pdf", dnew);
}
catch (Exception e)
{
   e.printStackTrace();
}
```

Another example that creates page views from a document created or loaded by other means, and prints without a *JWordTextPane*:

```
JWordEditorKit kit = new JWordEditorKit();

JWordDocument dnew = (JWordDocument) kit.createDefaultDocument();

...

// create view
Element root = dnew.getDefaultRootElement();
final DocumentView dv = new DocumentView(kit, root);
```

```
dv.recreate();
dv.setSize(dv.getWidth(), dv.getHeight());

// Print the contents
final PrinterJob job = PrinterJob.getPrinterJob();
PageFormat format = job.defaultPage();
PageSettings ps = dv.getPageSettings();
Paper pr = new Paper();
pr.setSize(Utils.point2Inch72(ps.getPageWidth()), Utils.point2Inch72(ps.getPageHeight()));
pr.setImageableArea(0, 0, Utils.point2Inch72(ps.getPageWidth()),
            Utils.point2Inch72(ps.getPageHeight()));
format.setPaper(pr);
job.defaultPage(format);

PrintHelper helper = new PrintHelper(format, dv);
job.setPageable(helper);
try
{
   job.print();
}
catch (PrinterException e)
{
   e.printStackTrace();
}
```

Mail Merge

The mail merge feature allows combining a data table with a document template to produce a document with a repeating content merged with data. You can use mail merge through the UI, by first defining a data table, then adding merge fields into the document template and then invoking the “merge” command to produce the final document.

The mail merge feature can be used programmatically too. Here is the list of JWordTextPane methods related to this operation:

Method	Description
getMergeData()	Returns the current merge table
getMergeColumns()	Returns the list of merge columns
insertMergeField(int, String)	İnserts a new merge field of the given column, into the given offset
setMergeData(Vector,Vector)	Sets document merge data. Data is a Vector of Vectors, column names is a Vector of Strings
setMergeData(Vector)	Sets document merge data. Data is a Vector of Vectors.
mergeDocument()	İnvoke document merging

This is an example use of merge methods. The merge data is gathered from some outer source and combined with the template document.

Programmatic data table definition in template editing phase:

```
//create merge columns
Vector<String> columns=new Vector<String>(20);
columns.add("id");
columns.add("name");
columns.add("surname");
columns.add("phone");
columns.add("creditLimit");
```

Merging the loaded template with data:

```
//create merge data
Vector<Vector> data=new Vector<Vector>();
//fill-in data from some data source (file, JDBC, network,..)
//here is a generated sample
for(int i=0;i<10;i++)
{
   Vector<String> rw=new Vector<String>(5);
   rw.add("id"+i);
   rw.add("name"+i);
   rw.add("surname"+i);
   rw.add("123456789"+i);
   rw.add(""+(i*1000+500));
   data.add(rw);
}
//merge document
editor.setMergeData(data);
editor.mergeDocument();
```

From this point on, you can save, print and edit the merged document, just like any other one.

Import / Export

The *JWordTextPane* class provides *cut*(), *copy*(), *paste*(), *pasteSpecial*(*String*) and *selectAll*() methods to interact with the sytem clipboard. The *cut*() and *copy*() methods work on the selected region in document. Paste methods paste clipboard data to the current insertion position. You can also use the *getSelectedTextAsRTF*() method to retrieve selected text as an RTF formatted String.

The mime types used for clipboard operations are defined in *JWordTransferHandler*:

MimeTXT	"text/plain"	plain text
MimeRTF	"text/rtf"	richtextformat
MimeWMF	"image/x-wmf"	windowsmetafile
MimeEMF	"image/x-emf"	enhancedwindowsmetafile
MimeIMG	"image/x-java-image"	image
MimeSVG	"image/svg"	SVG
MimeSTR	"application/x-java-jvm-local-objectref"	unicodetext

Editor Options

You can customize editors look and feel by setting some options. *JWordTextPane* provides *setOption*(*String*, *ListValue*) and *getOption*(*String*) methods and some shortcuts for this purpose.

The list of options available are:

Method	Description
OPT_UNIT	The display unit (pt,cm,mm,inch) Values: JWordTextPane.unitInch JWordTextPane.unitCM JWordTextPane.unitMM JWordTextPane.unitPT
OPT_DISPLAY_PAGE_BORDERS	Show page borders (yes,no) Values: JWordTextPane.optYES JWordTextPane.optNo
OPT_SHOW_HIDDEN_CHARS	Display hidden characters (yes,no) Values: JWordTextPane.optYES JWordTextPane.optNo
OPT_AUTO_SPELL_CHECK	Enables/disables automatic spell checker.

The default view mode can be altered by the *setViewMode(int,boolean)* method.

Example:

```
editor.setOption(JWordTextPane.OPT_UNIT, JWordTextPane.unitMM);//set units to milimeters
editor.setViewMode(JWordTextPane.VM_NORMAL, true); //normal view mode
```

Localization

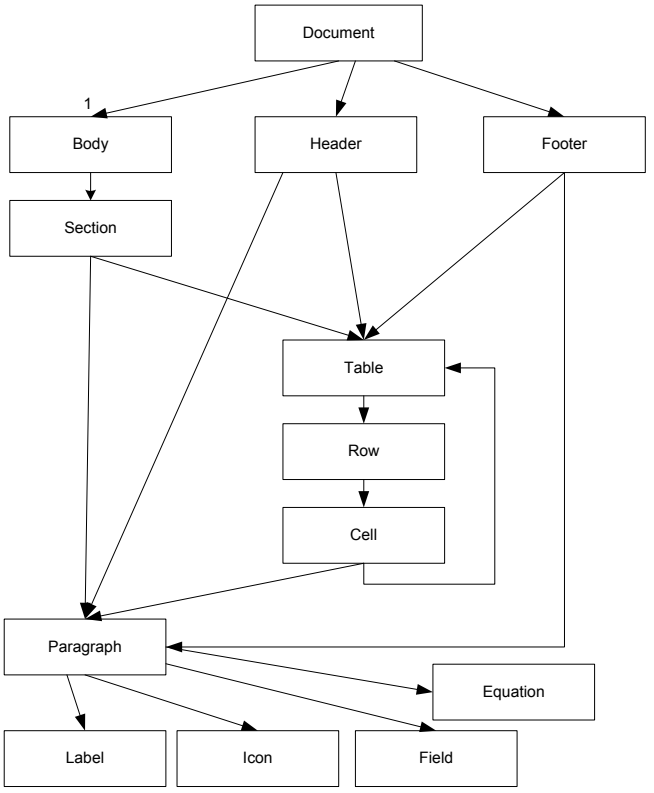
You can localize the user interface by providing translated version of the resource bundle. Take the resource bundle, *jword.properties*, from the JWord's jar file, and translate. In each row,a resource key is followed by an equality sign, which is followed by the translated text. Locale shortcut must be added to the property file name (such as "*jword_fr.properties*" for French translation). The file must be in the root of the application. You can also give a forced locale to the *JWordController* at the constructor which makes the controller use the given locale instead of the default one.

Licensing

By the default, JWord works in unlicensed mode which displays a notice text in the background and disables some features. In order to make it work in licensed mode, the *jword.lic.xml* file must be copied under the working directory of your application. You will receive this file upon purchase of the package. JWord will check this file in the initialization of the EditorKit. You can check your license details from the about box.

The Document Model

The document is composed of a single body and header-footer parts for each section. The body is divided into one or many sections. Each section, header and footer elements are composed of paragraph or table elements. Tables are constructed by rows, and rows are constructed by cells. Cell elements contain paragraphs or nested table elements. The leaf elements icon, label and field always belong to paragraph elements.



Attributes

The branch and leaf elements may have the following attributes

The attributes defined in the Text API and set through StyleConstants class:

Name	Description	Applies to
Alignment	Paragraph alignment	Paragraph
Background	Background color	Label, Field
BidiLevel	Bidirectional level of a character	Label, Field
Bold	Bold attribute	Label, Field
FontFamily	Font familiy	Label, Field
FirstLineIndent	First line indent (twips)	Paragraph
FontSize	Fon size (twips)	Label, Field
Foreground	Foreground (font) color	Label, Field
IconAttribute	Image icon	Icon
Italic	Italic attribute	Label, Field
LeftIndent	Left indent (twips)	Paragraph
LineSpacing	Line spacing value	Paragraph
NameAttribute	Name of element	All
Orientation	Paragraph orientation	Paragraph
ResolveAttribute	Name of parent attribute set (style)	All
RightIndent	Right indent (twips)	Paragraph
SpaceAbove	Space above (twips)	Paragraph
SpaceBelow	Space below (twips)	Paragraph
StrikeThrough	Strikethrough attribute	Label, Field
Subscript	Subscript attribute	Label, Field

Sperscript	Superscript attribute	Label, Field
TabSet	Tab set	Paragraph
Underline	Underline attribute	Label, Field

The attributes defined in JWord and set through StyleConstantsExt class:

Name	Description	Applies to
BackColor	Background color	Table, Row, Cell, Paragraph
Borders	Border thicness and color	Table, Row, Cell, Paragraph
Break	Soft break type	Paragraph
CellMargins	Cell margins (insets) in twips	Table, Cell
CellMerge	Cell merge attributes	Cell
Columns	Column definitions (width) for table	Table
ColumnSettings	Colum setting for section	Section
Field	Field type and settings	Field
HAlign	Horizontal alignment for table	Table
Height	Height in dimensions	Icon, Row
InheritInsets	Should cell inherit margins from table	Cell
LineSpaceMode	Line spacing mode (exact, atleast, multi)	Paragraph
Numbering	Number and bullet settings for a paragraph	Paragraph
PageSettings	Page size, orientation and margin settings	Document
PreventSingleLines	Prevent single lines option	Paragraph
PreventSplitting	Prevent splitting option	Paragraph
RowHeightMode	Row height mode for table rows (exact, atleast, none)	Row
StyleFollower	Name of the following style	Style
StyleType	Type of style (character, paragraph)	Style
TableLeftIndent	Left indent for tables (in twips)	Table
VAlign	Vertical alignment (top, bottom, center)	Cell, Icon
VerticalPosition	Vertical position wrt base-line (+/- in twips). Overrides sub, super attributes	Label, Field
Width	Width in dimensions	Table, Cell, Icon
Hyperlink	Hyperlink address	Label, Icon, Field, Equation
EquationData	MathML (XHTML) of equation	Equation
EquationAlign	Vertical alignment for equations (0.0-1.0)	Equation
CellSpacing	Spacing between table cells in twips	Table
BorderCollapse	Should neighboring borders be collapsed to one in the table	Table
CharSpacing	Character spacing (positive or negative).	Label