RapidSpell Desktop

v2.2.0 Java Edition

Software Component

User's Guide

Keyoti

Contents

Terms & Conditions

If you do not agree to these terms and conditions then you may not install, use, or distribute RapidSpell.

1. License and Ownership RapidSpell is a licensed software product. RapidSpell, including its accompanying files and documentation, is owned and copyrighted by Keyoti, © Copyright 2002, all rights reserved. Keyoti grants the user in possession of an official receipt of purchase a nonexclusive license to download and use the software, for any lawful commercial or non-commercial purposes provided the terms of this agreement are met.

2. Distribution If you purchased an Individual License Version of RapidSpell, you many copy and use the RapidSpell distribution file (.zip) on any systems you use but for your individual use only. Individual use is defined as one person having the ability to read or copy any of the files originally contained in the RapidSpell distribution file including (but not limited to) the .jar, .class, .java, .html, .gif, .jpg, .bat, and .txt files. You must take appropriate safeguards to protect this product from unauthorized access by others.

If you purchased a Site License Version of RapidSpell, then you may provide copies of the RapidSpell distribution file (.zip) to anyone employed by your company whose primary work address is within a 100 mile (160 km) radius of your primary work address. These individuals then have license rights and responsibilities equivalent to the Individual License Version. In all cases the copies of the RapidSpell distribution file (.zip) must be unaltered and complete, including this license agreement and all copyright notices.

You may not reverse engineer, disassemble, decompile, or modify this software or its accompanying documentation in any way.

You may reproduce and redistribute a copy of the class files in com\keyoti\rapidSpell as part of any Java-based software developed and licensed by you that itself uses RapidSpell provided
a) your software has clearly distinct and added functionality,
b) you are responsible for all technical support,
c) the RapidSpell application programming interfaces are not documented, exposed, or otherwise made available by your software,
d) attribution is given to Keyoti (http://www.keyoti.com) in any documentation or screen displays where other credits appear, and
e) you do not allow recipients of your software to reverse engineer, disassemble, decompile, or copy portions from your software allowing them to gain separate access to RapidSpell or any parts of it.

The source code samples included with this package and in the RapidSpell documentation may be freely used, modified, incorporated, and distributed without restriction as part of any software that uses RapidSpell itself.

3. Warranty Disclaimer and Limitation of Liability RapidSpell is licensed to the user on an "AS IS" basis. Keyoti MAKES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS SOFTWARE AND ITS ASSOCIATED FILES AND DOCUMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. KEYOTI DOES NOT WARRANT THAT THE OPERATION OF THIS SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE, OR THAT DEFECTS WILL BE CORRECTED. You the user are solely responsible for determining the appropriateness of this software for your use and accept full responsibility for all risks associated with its use. Keyoti is not and will not be liable for any direct, indirect, special or incidental damages in any amount including loss of profits or interruption of business however caused, even if Keyoti has been advised of the possibility of such damages. Keyoti retains the right to, in its sole discretion, refund the purchase price of this software as a complete and final resolution to any dispute.

SPECIFIC DISCLAIMER FOR HIGH-RISK ACTIVITIES.The SOFTWARE is not designed or intended for use in high-risk activities including, without restricting the generality of the foregoing, on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Keyoti and its suppliers specifically disclaim any express or implied warranty of fitness for such purposes or any other purposes.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES.To the maximum extent permitted by applicable laws, in no event shall Keyoti or its suppliers be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of information, or other pecuniary loss) arising out of the use of or inability to use this Keyoti product, even if Keyoti has been advised of the possibility of such damages. Because some states and jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

4. Support and Source Code Licensing Keyoti does not provide formal customer support for RapidSpell. However, we make every effort to ensure the quality of our products, and questions, bug reports, and feature requests are encouraged, please use email. Due to resource limitations, it may not be possible to resolve all issues and no assurances can be given as to when or if problems will be addressed. Customers incorporating RapidSpell into software for deployment or sale are encouraged to purchase the Source Code License Version of RapidSpell which will enable self-support, debugging, and modifications or extensions to the base functionality (but not distribution of the source code itself).

Introduction

Thank you for purchasing RapidSpell, please keep up to date through our website keyoti.com and feel invited to post feedback. We also ask that you register your purchase so that we may offer you support and inform you of free product upgrades from time to time, please go to

http://keyoti.com/care/register

Overview

RapidSpell consists of 2 JavaBean spelling components (GUI and non-GUI) which provide for rapid integration within your application. Keyoti has developed these highly granular components with simplicity of application development in mind.

The Jar holds a graphical user interface Bean, which can be implemented with 2 lines of code to supply complete spell checker functionality to any Swing based application and a non-GUI Bean which provides a clean API for model (business/logic) level integration (notably of use with web applications). Whilst simplicity of use has been a major guiding force in the component design, user customisation has not been spared, the GUI can be easily used as a base class and extended to provide UI customisation and the non-GUI component provides access to core spell checking functionality.

Common Uses

The GUI Bean can check any JTextComponent (eg; JEditorPane, JTextPane, JTextArea, JTextField) component including your own sub-classes of JTextComponent -allowing for uses such as:

Word Processor & DTP applications
Email programs
Web content management applications /WYSIWYG editors
Database forms
Internal company messaging applications
Any GUI application that allows English text input!

The non-GUI Bean provides core spell checking functionality, allowing it to be useful in any application, including:

Server applications
Console based applications
AWT GUI components

Requirements

The components were built to JDK1.2.2 specification and are expected to work with all JDKs from 1.2.2 up.

Improvements In v2

v2.2
RapidSpellGUIDialog class added, extends JDialog instead of JFrame like RapidSpellGUI
New memory optimization properties added
Improved as you type performance in large documents
It is now possible to customize the popup suggestions menu, and to control underline painting
Improved suggestions for short (2 or 3 letter) words
XML/HTML support added to RapidSpellAsYouType
v2.1
Improvement in multiple text box checking
Added as you type spell checking
Improvement in word lookup check speed
Improvement in phonetic suggestions speed
New alternative 'hashing' suggestions method added, runs upto 40x faster than phonetic suggestions
New methods for behaviour customization
More accurate dictionaries
New Dict dictionary files allow runtime dictionary swapping
XML/HTML support
Built-in UI texts for French, German, Italian and Spanish

The Dictionaries

Included are 3 dictionary assemblies, UK and US English dictionaries, each containing ~110K words, and a combined UK and US dictionary containing ~120K words. These can be found in the 'dictionaries' directory. With v2 a new dictionary format has been added. These dictionaries reside in *.dict files and have been added to improve flexibility, for more information please see the Dict Files section.

Installation

Windows users can double click the RapidSpell.jar icon to see an example application. To install RapidSpell so that you may use it in your application development requires two steps, you must add the RapidSpell Jar file 'RapidSpell.jar' to your project class path and **put 'RapidSpellMDict.jar' in the same directory as 'RapidSpell.jar'**. Exactly how you do this depends on your development environment;

For the command line compiler

1. Copy the jar files to a convenient directory eg; c:\jars
2. Specify the class path explicitly in your javac command
   eg; **javac -classpath %classpath%;c:\jars\RapidSpell.jar MyApplication.java**
3. Specify the class path explicitly when running your application
   eg; **java -cp %classpath%;c:\jars\RapidSpell.jar MyApplication**

For the Ant build tool

1. Copy RapidSpell.jar to the lib directory off of your project directory.
2. Modify your build.xml file to add the Jar to your class path, to do this find the compile target:
   eg:
   <target name="compile" depends="init">
   .....
   </target>
   Within the compile target tags are 'javac' tags which can contain 'classpath' tags, inside the class path tags add this tag
   <pathelement location="lib/RapidSpell.jar"/>
   Which will give you something like the following;

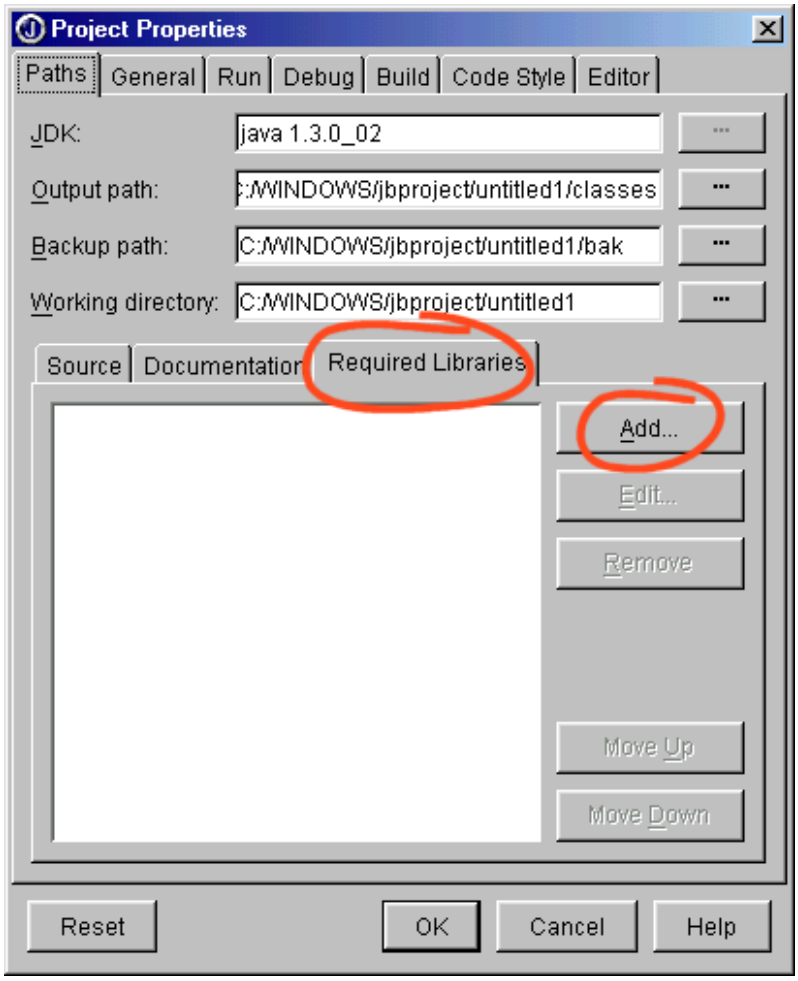
```
<target name="compile" depends="init">
 <copy todir="${build}">
   <fileset dir="${etc}"/>
 </copy>
  <javac srcdir="${src}" destdir="${build}">
   <classpath>
    <pathelement path="${classpath}"/>
 <pathelement location="lib/RapidSpell.jar"/>
   </classpath>
   </javac>
 </target>
```

3. Modify your run.bat script to include the lib\RapidSpell.jar file in your class path
   eg: java -cp dist\lib\MyApp.jar;**lib\RapidSpell.jar** MyApp

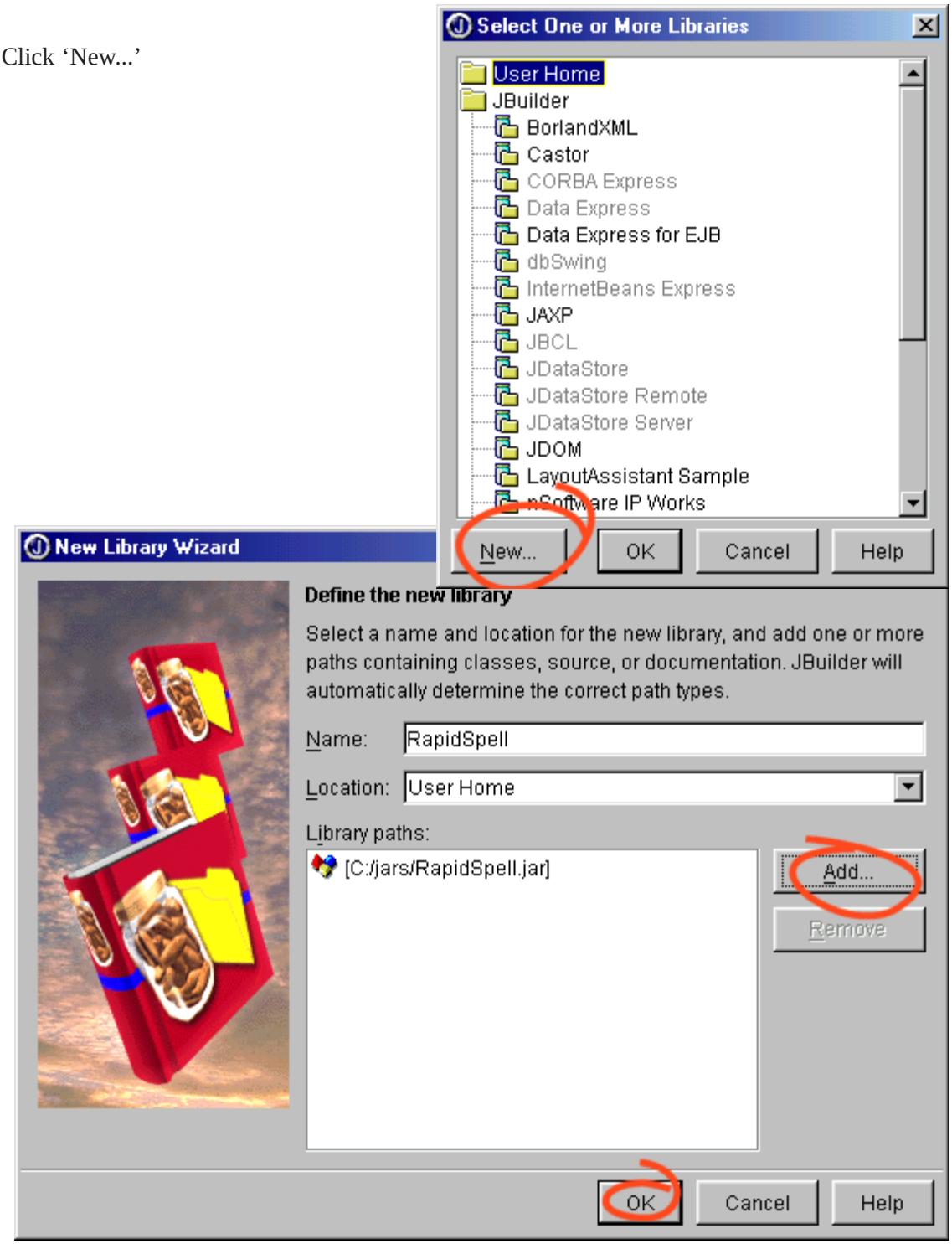
For the JBuilder IDE

1. Open your project and go to the Project menu, select 'Properties...'

2. In the 'Required Libraries' tab click 'Add'.



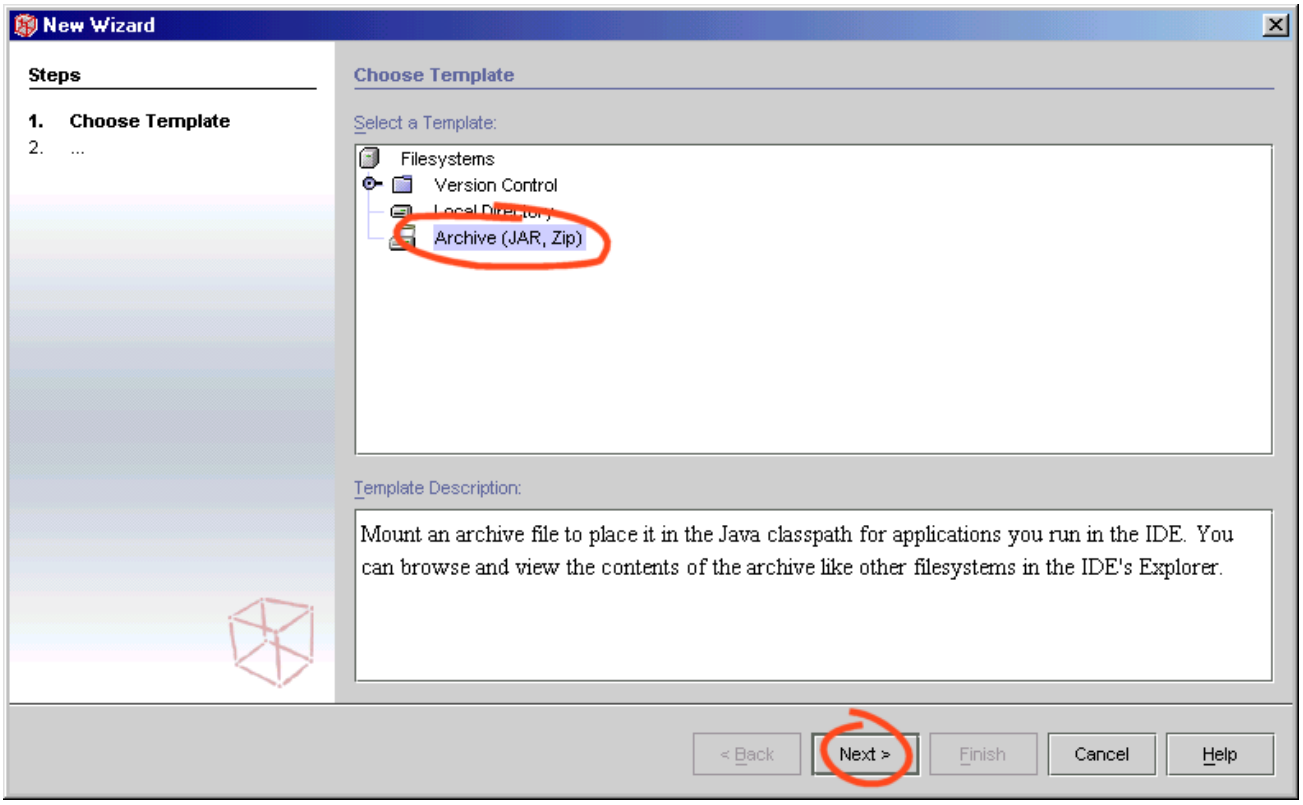
3. Click 'New...'



4. Name the new library RapidSpell and click 'Add' to enter the path of the RapidSpell.jar file. Click'Ok' in every window until you are back in the IDE main window.

For the NetBeans IDE

1. Mount the RapidSpell.jar file by going to the 'File' menu and selecting 'Mount Filesystem', select 'Archive (Jar/Zip)' from the 'Choose Template' pane, click next and enter the path to your RapidSpell.jar file.



2. In the NetBeans 'Explorer' pane navigate to the RapidSpell.jar file and expand it, expand com, keyoti, and rapidSpell folders. Right click on the RapidSpellGUI bean select 'Tools'->'Add to Component Palette' then choose an appropriate component palette and click ok. This Bean will now be available on the palette you selected.

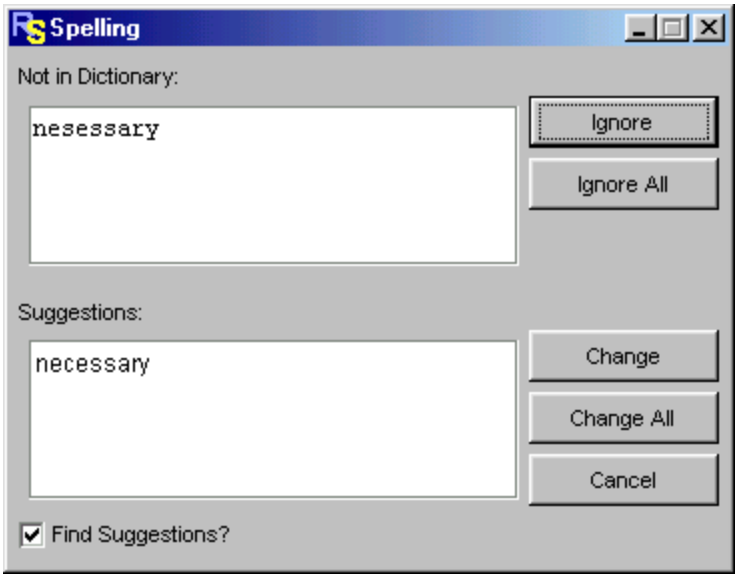
For all IDEs

The principles in all the installation instructions outlined here are the same, put the RapidSpell.jar and RapidSpellMDict.jar files somewhere convenient and then add RapidSpell.jar to your project class path, your IDE will have detailed instructions on how to do this.

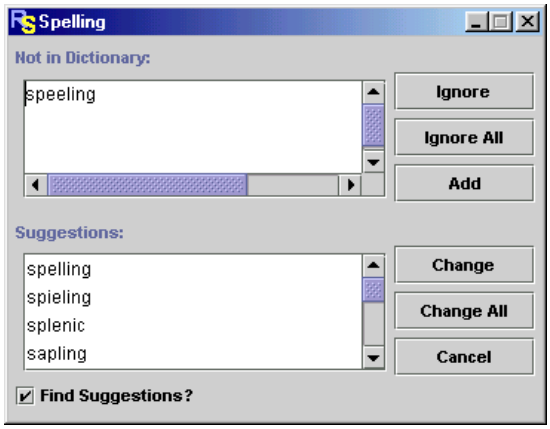
Once installed on your class path you are free to *import* the API in your project classes.

The Beans

RapidSpellGUI & RapidSpellGUIDialog



Looks Great In Any Look & Feel



Default Look & Feel
Window icon can be changed

RapidSpellGUIDialog is a JDialog based version of RapidSpellGUI - from here on anything applicable to RapidSpellGUI is also applicable to RapidSpellGUIDialog. RapidSpellGUI is a 100% customisable UI, provides user dictionary functionality, is multi-threaded for slick usability and is extremely simple to use. The interface has a familiar layout and functionality.

RapidSpellGUI will interactively check any JTextComponent derived class, this includes JEditorPane, JTextPane, JTextArea, JTextField and their sub-classes! When invoked in an application the Bean will query the user on each misspelt word allowing the following functions;

Ignore
Ignore All occurrences of misspelt word
Add to user dictionary if provided
Change misspelt word to suggestion or user amended text
Change All occurrences of misspelt word
Cancel

The GUI checks selections (querying to continue when finished), wraps from current caret position back to current caret position and handles the user editing the text in your application while checking is in progress.

The core RapidSpellChecker has tested at approximately **50,000 words/second** on a standard 1GHz machine.

RapidSpellGUI Summary

Here is a summary of the main features of the RapidSpellGUI Bean, please consult the rest of the documentation for more on the core spell checker engine component, RapidSpellChecker.

Incorrectly spelt words in the JTextComponent being checked are highlighted as the user is queried.
RapidSpellGUI allows the user to manually correct words in either the query box or the JTextComponent being checked.
Multiple JTextComponents can be checked seamlessly by deactivating the 'finished' message box and cycling through all components with check().
Multi-threaded.
Fully customizable user-dictionary at development and install time.
Checker wraps past end of document.
Checks selections and queries user on whether to continue.
Fully customizable GUI layout.
Built-in switchable language GUI texts

Key API Points

Methods

RapidSpellGUI();	-argumentless constructor
buildGUI();	-overrideable GUI layout method
setUserDictionaryFile(File file);	-uses a file as a user dictionary in addition to main dictionary
check(JTextComponent c);	-open GUI window and spell check text in any JTextComponent
setDictFilePath(String filePath);	-sets the Dict dictionary file, for main dictionary use
setCheckerEngine(ICheckerEngine e);	-sets the core spell checker object, allows customization
setSuggestionsMethod(String m);	-sets whether to use hashing or phonetic suggestions
setSeparateHyphenWords(boolean b)	-sets hyphenated word behaviour
setLookIntoHyphenatedText(boolean b)	-sets hyphenated word behaviour
setIgnoreXML(boolean b)	-sets XML/HTML behaviour
setGUILanguage(LanguageType l)	-sets the GUI text language

Fields

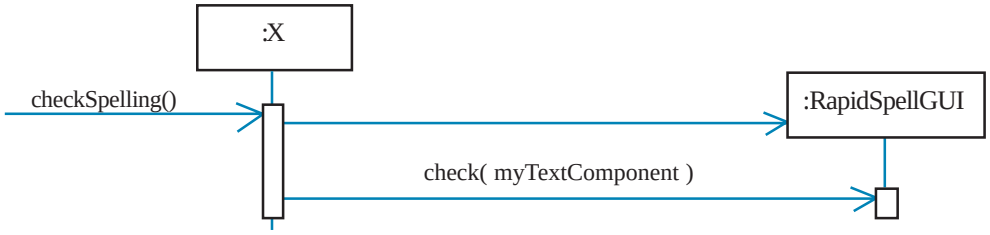
Protected GUI objects allow sub-classes to use in their own designs.

Events

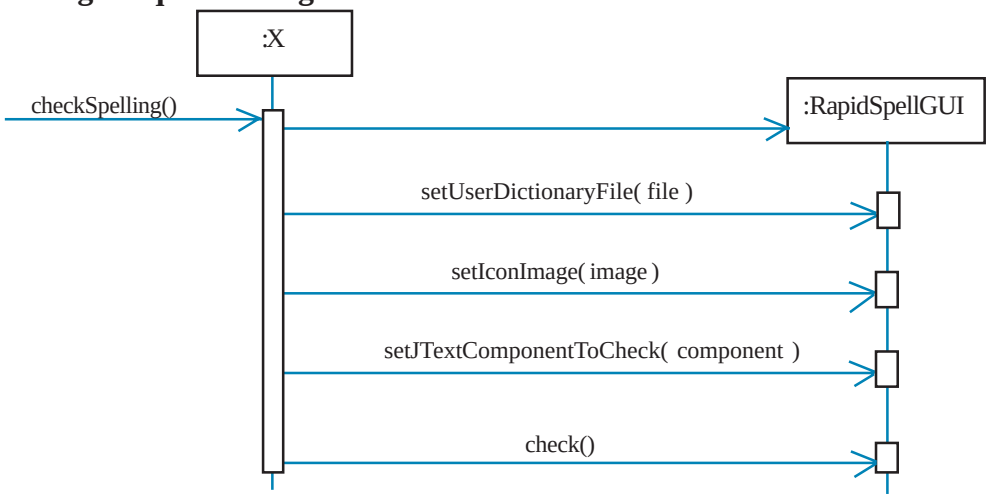
SpellCheckStarted and SpellCheckFinished events.

Please consult the API docs for detailed information and later sections for details on customising the UI.

Simplest Usage Sequence Diagram

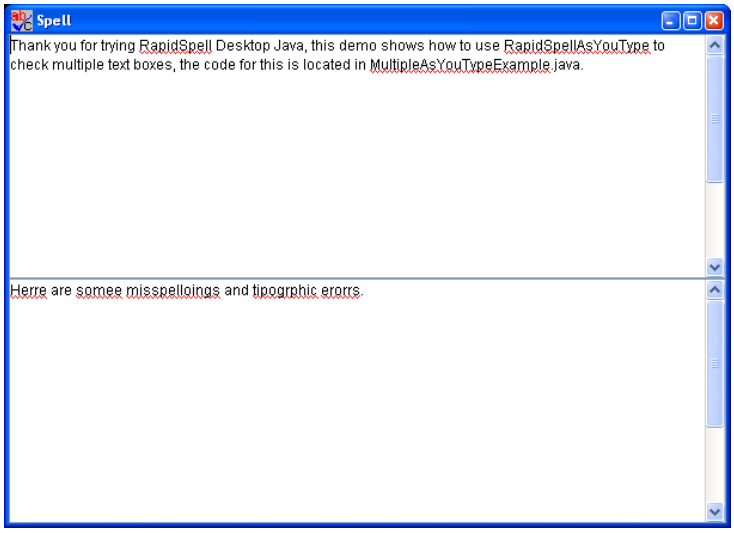


Detailed Usage Sequence Diagram



RapidSpellAsYouType

Provides as-you-type spell checking to JTextComponent and it's derivative classes (JTextPane, JEditorPane etc). RapidSpellAsYouType is extremely simple to use, and highly configurable - it provides full in-document highlighting and monitoring, so that, if the user edits a word and it becomes misspelled, or if the user pastes some text with misspellings, those errors are displayed with an underline.



RapidSpellAsYouType Summary

Here is a summary of the main features of the RapidSpellAsYouType Bean.

Incorrectly spelt words in the JTextComponent are underlined in a developer configurable underline.
Right clicking on an underline produces suggestions context menu - with 'Ignore All' and 'Add' to dictionary options.
Multiple JTextComponents can be checked seamlessly by setting setTextComponents(JTextComponent[] textBoxArray).
Multi-threaded.
Fully customizable user-dictionary at development and install time.
All document changes are correctly monitored and underlines tracked.
Built-in switchable language GUI texts.

Members

RapidSpellAsYouType is as configurable as RapidSpellGUI, please see the API docs for all methods and properties available.

RapidSpellChecker

Provides a model (business/logic) level spell checker API which can be used in a client-server (eg; web, intranet) environment and any applications where a client GUI does not exist. RapidSpellChecker has been benchmarked at checking 50,000 words/second on a standard 1GHz PC. User dictionaries can optionally be used, allowing words to be added. This component can be used in 2 different ways, in an iterative fashion or on a query by query basis. Please see the API docs for code level details.

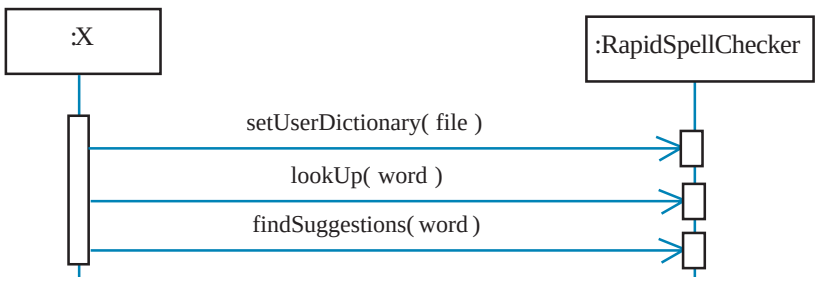
Using RapidSpellChecker in an iterative manner provides complete textual correction of strings similar to a GUI based spell checker. A string is loaded into RapidSpellChecker and then successive calls to the nextBadWord() method move through the string allowing findSuggestions() and changeBadWord(replacement) to be called to find spelling suggestions and optionally correct the misspelt words in the text, finally getAmendedText() returns the corrected text.

Alternatively RapidSpellChecker may be used in a simple query by query basis, calls to lookUp(word) and findSuggestions(word) methods allow bare bones spell checking functionality.

Key API Points

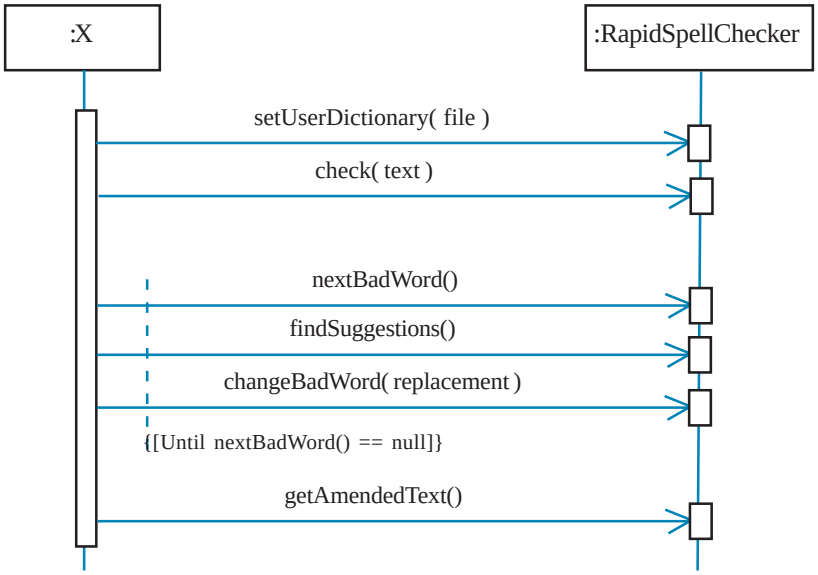
Methods	
RapidSpellChecker()	-argumentless constructor
check(text [, startPosition])	-begins checking text
findSuggestions(word)	-finds suggestions for word
lookUp(word)	-checks if word is in either main or user dictionaries
setUserDictionary(file)	-allows you to define which file will be used as a user dictionary
ignoreAll(word)	-ignores all future occurrences of word
addWord(word)	-adds word to the user dictionary if it exists
nextBadWord()	-finds the next misspelt word in the text
changeBadWord(newWord)	-changes the misspelt word last identified by nextBadWord()
getAmendedText()	-returns the corrected text

Query by Query Usage Sequence Diagram



Iterative Usage Sequence Diagram

See use cases section for examples.



Dict Files

RapidSpell now supports it's own proprietary dictionary format, these dictionaries are called Dict files. They have been developed to allow more flexibility than the existing dictionary format. The existing MDict jar format will continue to be supported, the advantages and disadvantages of the formats are outlined below:

New Dict File;

Advantages
- Swappable during run-time (necessary for other languages)
- Customizable by developer (future)

Disadvantages
- Larger file size

Existing MDict.DLL format;

Advantages
- Compressed

Disadvantages
- Slower

If it is necessary to minimise the dictionary size then the Jar format should be used, however for all other uses the new Dict File format is preferable.

Using Dict Files

The older MDict.Jar type file must reside in the same directory as the main RapidSpell Jar, it will be used if the DictFilePath property of RapidSpellGUI is null. To use a Dict file simply set the DictFilePath property to the path of the Dict file to be used, this should be done BEFORE the Check method is called, to prevent redundent dictionary loading.

Dictionary Customization

It is possible to customize the dictionaries and create new ones using the convenient Dict Manager tool included in the full version of RapidSpell Desktop.

Use Cases

Below are some use scenarios of the 2 spell checker components supplied, please refer to the API documentation for additional detail.

Simple GUI Text Editor

To use the GUI spell checker Bean as an integral part of your application simply instantiate it, set the JTextComponent you wish this GUI to spell check and call check(), which will bring up this component in a new JFrame allowing the user to correct their document in the JTextComponent.

Example, note this code is highly simplistic for demonstration purposes, the main window does not even contain a close window event handler.

Code Example

```
import com.keyoti.rapidSpell.*;
import com.keyoti.rapidSpell.desktop.*;
import javax.swing.*;
import java.awt.event.*;

public class TestGUI extends JFrame {

        JTextArea box = new JTextArea("This is sume text to check.", 20, 60);
        RapidSpellGUI rapidGUI = new RapidSpellGUI();

        public TestGUI() {
                //put the text box in the JFrame
                getContentPane().add(box);

                //pack and display the JFrame
                pack();
                setVisible(true);

                //open the GUI spell checker and begin checking the text box.
                rapidGUI.check( box );
        }

        public static void main(String[] args) {
                TestGUI t = new TestGUI();
        }
}
```

Instead of this call;

```
rapidGUI.check(box);
```

We could have made 2 calls;

```
rapidGUI.setJTextComponentToCheck(box);
rapidGUI.check();
```

This is typically the format taken when designing with RAD tools.

Custom RapidSpellGUI Interface

You can easily extend the RapidSpellGUI class to customise the GUI design, preserving all the functionality of the original GUI. To do so the only contract is that your sub-class overrides buildGUI(), calls getContentPane().add(X) to add your design to the JFrame and uses these protected fields as GUI components;

JButton **addButton** Function button - adds word in the query pane to user dictionary if specified.
JButton **cancelButton** Function button - cancels the search and disposes of this JFrame.
JButton **changeButton** Function button - changes current misspelt word in document to selected suggestion or amended query pane word.
JButton **changeAllButton** Function button - changes every occurrence of current misspelt word in document.
JButton **ignoreAllButton** Function button - ignores all occurrences of current misspelt (bad) word.
JButton **ignoreButton** Function button - ignores current bad word and continues checking document.
JTextArea **queryWordPane** JTextArea where not in dictionary (bad words) are queried and replacements can be entered.
JList **suggestionsList** JList where suggestions are loaded
JCheckBox **suggestionFinderCheckBox** Check box - enables/disables the suggestion finder.
JLabel **notInDictionaryLabel** JLabel marking the not in dictionary query pane.
JLabel **suggestionsLabel** JLabel marking the suggestions JList, is changed to "Finding Suggestions..." periodically.
boolean **findSuggestions** Whether this should look up suggestions for misspelt words or not.

A very simple example would be the following code, which can be called instead of RapidSpellGUI but used in the same manner.

Code Example

```
import com.keyoti.rapidSpell.desktop.RapidSpellGUI;
import javax.swing.*;
import java.awt.*;

public class CustomGUI extends RapidSpellGUI
{
	public CustomGUI()
	{
		super();
	}

	// VERY SIMPLE, BUT UGLY LOOKING EXAMPLE
	public void buildGUI()
	{
		Box myBox = Box.createVerticalBox();

		//define the protected fields as I want
		ignoreButton.setText("Ignore");
		ignoreAllButton.setText("Ignore All");
		changeButton.setText("Change");
		changeAllButton.setText("Change All");
		cancelButton.setText("Cancel");

		//add all the protected fields I want to use and layout as I wish
		myBox.add(notInDictionaryLabel);
		myBox.add(queryWordPane);
		myBox.add(ignoreButton);
		myBox.add(ignoreAllButton);
		myBox.add(new JScrollPane(suggestionsList,
			JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
			JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED ));
		myBox.add(changeButton);
		myBox.add(changeAllButton);
		myBox.add(cancelButton);

		getContentPane().add(myBox);
	}
}
```

Notice that the protected fields of RapidSpellGUI were used with out declaration and without adding any listeners to them.

As You Type Checking

The built in as you type class RapidSpellAsYouType makes as you type checking on JTextComponents very simple - the minimum amount of code to write is two lines, one to instantiate the RapidSpellAsYouType class and one to set the text component to work with.

Code Example Multiple As You Type Checking

```
import com.keyoti.rapidSpell.desktop.*;
import com.keyoti.rapidSpell.event.*;

import javax.swing.*;
import javax.swing.text.JTextComponent;
import java.awt.event.*;
import java.awt.Dimension;
import java.awt.BorderLayout;
import java.io.File;

/** Demonstrates how to use RapidSpellAsYouType to check more than one
JTextComponent */
public class MultipleAsYouTypeExample extends JFrame{

      JTextArea box1 = new JTextArea("Thank you for trying RapidSpell Desktop
Java, this demo shows how to use RapidSpellAsYouType to check multiple text
boxes, the code for this is located in MultipleAsYouTypeExample.java. \n", 20,
60),
      box2 = new JTextArea("Herre are somee misspelloings and tipogrphic
erorrs.", 20, 60);

      RapidSpellAsYouType rapidAYT;

      public MultipleAsYouTypeExample() {
            super("Spell");

            addWindowListener(new WindowAdapter() {
                  public void windowClosing(WindowEvent e) {
                        System.exit(0);}});

            getContentPane().setLayout(new BorderLayout());

            box1.setLineWrap(true);
            box1.setWrapStyleWord(true);

            box2.setLineWrap(true);
            box2.setWrapStyleWord(true);
```

```
            JScrollPane scrollPane = new JScrollPane(box1);
            scrollPane.setPreferredSize(new Dimension(600, 205));
            getContentPane().add(scrollPane, BorderLayout.NORTH);

            JScrollPane scrollPane2 = new JScrollPane(box2);
            scrollPane2.setPreferredSize(new Dimension(600, 205));
            getContentPane().add(scrollPane2, BorderLayout.SOUTH);

            //Create as you type checker object
            rapidAYT = new RapidSpellAsYouType();
            rapidAYT.setTextComponents( new JTextComponent[]{box1, box2} );


            pack();
            setVisible(true);


      }


      public static void main(String[] args) {
            MultipleAsYouTypeExample t = new MultipleAsYouTypeExample();
      }


}
```

Non-GUI Spell Checker

The RapidSpellChecker Bean component is of particular use in non-GUI scenarios such as the web. Below is a very simple excerpt of how this component can be used.

Code Example

```
.........
RapidSpellChecker c = new RapidSpellChecker();
BadWord badWord;
Enumeration suggestions;

//check some text.
c.check("This is sume text.");

//iterate through all bad words in the text.
while((badWord = c.nextBadWord())!=null){

      System.out.println(badWord.getWord() + "- is not spelt correctly. Sugges-
tions:");

      try{
            //get suggestions for the current bad word.
            suggestions = c.findSuggestions().elements();

            //display all suggestions.
            while(suggestions.hasMoreElements()) {
                  System.out.println(suggestions.nextElement());
            }

            //change the bad word in the text with "replacement".
            c.changeBadWord("replacement");

      } catch (NoCurrentBadWordException e){
            System.err.println(e);
      }
}
System.out.println(c.getAmendedText());
.........
```

Advanced Topics

Memory Management

The Java garbage collector generally does a good job of memory management, in that, *eventually* used memory will be freed as objects are no longer used. However it is often desirable to explicitly free memory used for spell checking, for example when MDI child frames are closed.

RapidSpellGUI & RapidSpellGUIDialog

A bug (http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4726458) in most SDK versions prevents the GC properly disposing of unused frames - and since RapidSpellGUI maintains a reference to RapidSpellChecker (which holds the dictionary) the reference should be explicitly removed manually. When RapidSpellGUI or RapidSpellGUIDialog is finished with, and the user form is closed it is advisable to free up memory through

```
gui.dispose();
gui = null;
```

WeakReferences are used between the checker and the text component.

RapidSpellAsYouType

To free up resources used by RapidSpellAsYouType call

```
rapidSpellAsYouType.dispose();
rapidSpellAsYouType = null;
```

This will also unattach the spell checker from the text box (ie. remove underlines).

Suggestions Algorithm

It is possible to choose (programmatically) the algoritm used to find suggestions for spelling errors, there are two algorithms on offer, 'hashing' (default) and 'phonetic'. Each algorithm has it's own advantages and disadvantages;

Hashing (default): Very fast, roughly 40 times faster than the phonetic algorithm, works best on typographic errors and common spelling mistakes. Similar to most spell checker suggestions.

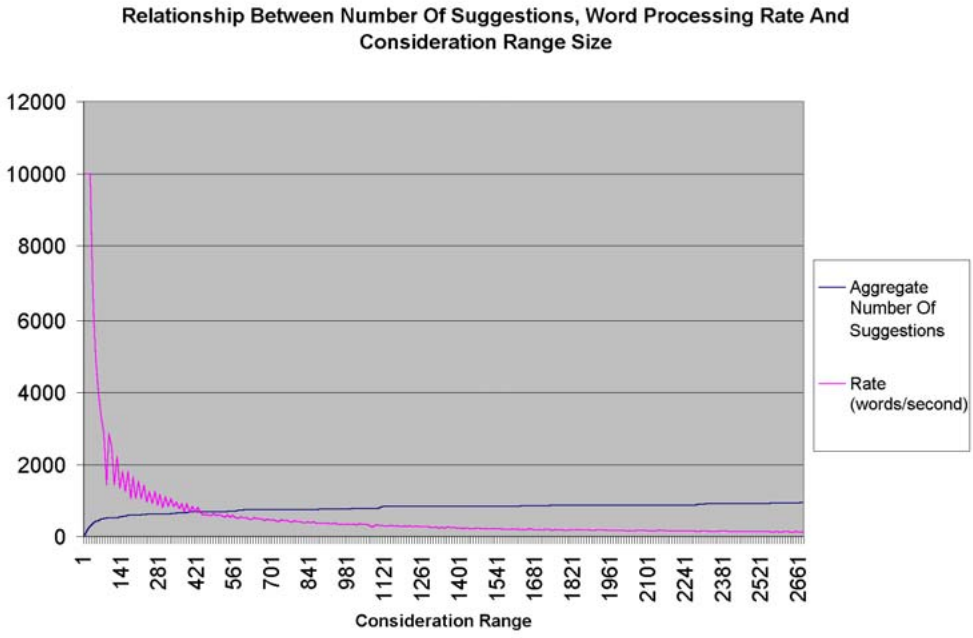
Phonetic (Sounds like, English only): Slower, works well with genuine attempts to spell correctly, works poorly with typographics errors. Can appear to produce wild results, although results do sound like the word in question. Does not work with non English languages. Unusual.

It is recommended that you choose the suggestions algorithm in line with your target audience. The phonetic algorithm, although appearing worse overall, is more suited for people with less experience of the English language, e.g. non native English speakers and children.

Consideration Range

The hashing suggestion algorithm is very fast because it accurately cuts down the list of words to look

at as possible suggestions. The factory default consideration range is 80, which provides optimal speed with a loss of only a fraction of possible suggestions. It is possible to set the consideration range through the RapidSpellGUI and RapidSpellChecker classes. The relationship between speed, number of suggestions and consideration range is shown below.



International UI & Dictionaries

RapidSpellGUI features built in text for several languages, the list of which will increase with subsequent releases of RapidSpell Desktop. Please consult the API docs for currently supported UI languages. To set the GUI text language use the setGUILanguage method, which takes a LanguageType static field identifier. Eg;

```
rapidSpellGUI.setGUILanguage(LanguageType.FRENCH);
```

To change the dictionary being used as a main dictionary either substitute the RapidSpellMDict.jar file for the other language (available for purchase through keyoti.com) or set the DictFilePath property to the location of the other language Dict file. Eg;

```
rapidSpellGUI.setDictFilePath("path/to/french.dict");
```

It is important that the 'language parser' for the spell checker matches the language being checked, for example in English an apostrophe is used as part of a word (eg. O'Mally, don't) but in French an apostrophe joins words (eg. j'aime). For this reason the language parser should be set to the type that best matches the language being checked. Eg;

```
rapidSpellGUI.setLanguageParser(LanguageType.FRENCH);
```

Although the RapidSpellGUI methods have been used in examples here these methods are reflected in the RapidSpellChecker nonGUI class, and usage is the same.

Conclusion

Thank you for using the RapidSpell component, we hope that you experience much success with it. Keyoti is committed to improving all of it's products and truly values customer feedback of any kind. If you would like to contact us about any of our products please do so through our web site.

http://www.keyoti.com/contact.html

We thrive on suggestions for components, if you have a component wish please don't hesitate to get in touch, it may be the next thing we develop!