# Creating and Editing Java Source Code

Using NetBeans IDE 3.6
Previous - TOC - Next

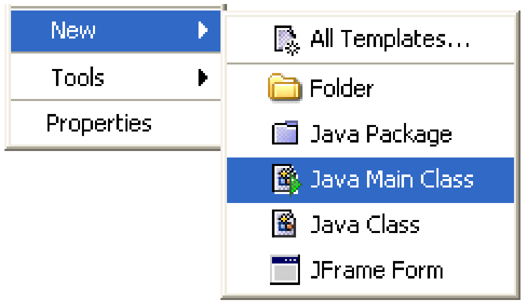
Creating and editing Java source code is the most important function that the IDE serves. After all, that's probably what you spend most of your day doing. NetBeans IDE provides a wide range of tools that can compliment any developer's personal style, whether you prefer to code everything by hand or want the IDE to generate large chunks of code for you.

This section covers the following topics:

- **Creating Java files** - Using the New wizard and the IDE's templates to create new files, GUI form templates versus Java source templates.
- **Editing Java files in the Source Editor** - Using code completion and abbreviations, generating bean properties and event listeners, working with import statements, search and selection tools, and formatting Java code.
- **Navigating between documents** - switching between open files, cloning the view of a file, and splitting the Source Editor.
- **Configuring the Source Editor** - customizing the Source Editor to fit your development style.

## Creating Java Files

NetBeans IDE contains templates and wizards that you can use to create all kinds of source files, from Java source files to XML documents and resource bundles.



The easiest way to create a file is to right-click the directory node in the Filesystem window where you want to create the file and choose from the New submenu in the node's contextual menu. The New submenu contains shortcuts to commonly-used templates and an All Templates command that you can use to choose from all NetBeans templates.

To demonstrate some of the IDE's source creation and editing features, let's recreate the `ColorPreview` class that comes with the `colorpicker` example in the IDE's sample code. Right-click any directory in your mounted filesystems and choose New > Java Class. Name the file `ColorPreview` and click Finish. The file opens in the Source Editor.

### GUI Templates and Java Templates

If you want to visually edit a Java GUI form using the Form Editor, you have to create the form's source file using the IDE's Java GUI Forms templates. This template group contains templates for AWT and SWING forms. For example, you cannot create a normal Java class file and then change it to extend `JPanel` and edit it in the Form Editor. The form must be created from the `JPanel` template.

## Editing Java Files in the Source Editor

The Source Editor is your main tool for editing source code. It provides a wide range of features that make writing code simpler and quicker, like code completion, highlighting of compilation errors, syntax highlighting of code elements, and advanced formatting and search features.

Although we talk about the Source Editor as one component, it is really a collection of editors. Each type of source file has its own editor that provides different functionality. In this section we'll be dealing with the Java editor, but many of the same concepts apply to other editors.

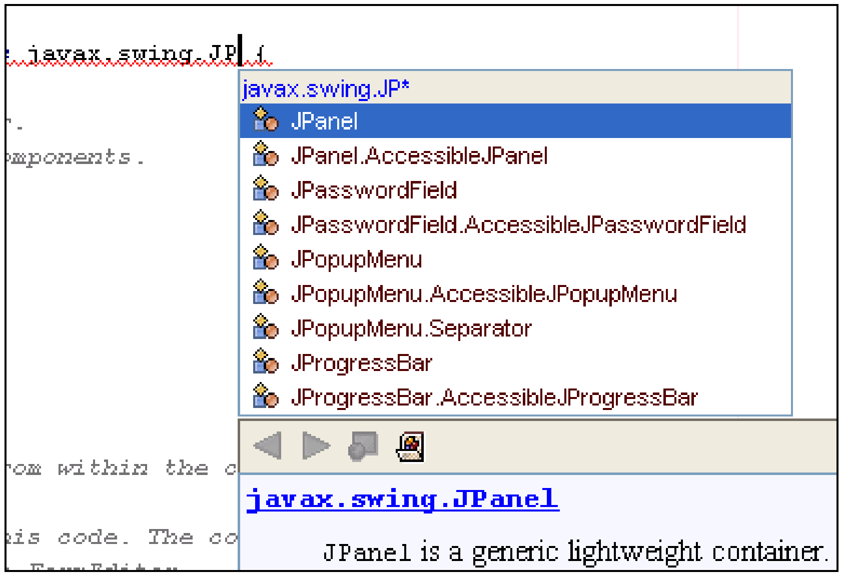
To open a Java source file in the Source Editor, double-click the file's node in the Filesystems window.

*Note: Double-clicking a Java form node (▣) in the Filesystems opens two tabs in the Source Editor: a source tab containing the Java source code for the form, and a Form Editor tab showing the design-time view of the form. To edit the source code for a Java form without opening the Form Editor, right-click its node and choose Edit.*

### Using Abbreviations, Word Matching, and Code Completion

The Source Editor provides many features that spare you from having to enter long Java class names and expressions by hand. The most commonly used of these features are abbreviations, code completion, and word matching.

Code completion in the Java Source Editor lets you type a few characters and then choose from a list of possible classes, methods, variables, and so on to automatically complete the expression. The Source Editor also includes a Javadoc preview window that displays the Javadoc documentation for the current selection in the code completion box, if any exists. The Javadoc is drawn from the compiled source files mounted in the IDE.


javax.swing.JP {
javax.swing.JP*
JPanel
JPanel.AccessibleJPanel
JPasswordField
JPasswordField.AccessibleJPasswordField
JPopupMenu
JPopupMenu.AccessibleJPopupMenu
JPopupMenu.Separator
JProgressBar
JProgressBar.AccessibleJProgressBar
javax.swing.JPanel
JPanel is a generic lightweight container.

Abbreviations are short groups of characters that expand into a full word or phrase when you press the space bar. For example, if you enter `psfs` and press the space bar, it expands into `public static final String`. For a full list of the IDE's default abbreviations, click **here**.

You can also add your own custom abbreviations for each type of editor. In the Options window, select Editing > Editor Settings > Java Editor and open the property editor for the Abbreviations property. You can use the Abbreviations property editor to add, remove, and edit the abbreviations for Java files.

Word matching is a feature that lets you type a few characters of a word that appears elsewhere in your code and then have the Source Editor generate the rest of the word. Type a few characters and press Ctrl-L to generate the next matching word or Ctrl-K to generate the previous matching word.

As a quick exercise, let's make `ColorPreview` extend `JPanel`. Put the insertion point after `ColorPicker` in the class declaration, then type `ex` and press the space bar to expand the abbreviation into `extends`. Then type the first few letters of `javax`. The code completion box should pop up after a few seconds. If it does not, you can always manually open it by pressing Ctrl-Space. Use the code completion box to enter `javax.swing.JPanel`.

## Configuring Code Completion

The IDE maintains a code completion database which it uses to provide suggestions for code completion and other features. The code completion database contains classes from the J2SK version 1.4, other commonly used APIs like the Servlet and XML APIs, and the sources in all of the filesystems you have mounted in your project. Whenever you mount a filesystem, the IDE automatically adds all of the filesystem's public and protected classes to the project's code completion database. You can also right-click the filesystem and choose Tools > Update Code Completions to configure which of the filesystem's classes are available for code completion.

In the Options window, you can disable and enable code completion and set the length of the pause before the code completion box appears in the Source Editor. Select Editing > Editor Settings > Java Editor and set the Auto Popup Completion Window property and the Delay of Completion Window Auto Popup property accordingly.

You can also turn off the Javadoc preview box for code completion. Select Java Editor and uncheck the Auto Popup Javadoc Window property.

## Adding Fields, Bean Properties, and Event Listeners

Even if you prefer to write your code the old-fashioned way, the NetBeans Java editor has some cool code generation features that you may find handy, especially when dealing with bean properties and event listeners.

Let's start by adding some of the fields for our colors in `ColorPreview`. Go to the first line after the class declaration and type in the following code:

```
    private int red;
```

Now let's turn this ordinary field into a bean property by making some getter and setter methods for it. Right-click anywhere in the field declaration and choose Tools > Generate R/W Property for Field. The following code is generated in the file:

```
    public int getRed() {
        return red;
    }

    public void setRed(int red) {
        this.red = red;
    }
```

The methods now show up under the Methods node. The Bean Patterns node now also contains a bean property node for `red`.

Now let's add both the field and the get and set methods at the same time. In the Filesystems window, right-click the Bean Patterns node for `ColorPreview` and choose Add > Property. In the dialog, enter `green` for the name and `int` for the type, then check Generate Field, Generate Get Method, and Generate Set Method and click OK. The following code is added to the file:

```
    private int green;

    public int getGreen() {
        return this.green;
    }

    public void setGreen(int green) {
        this.green = green;
    }
```

So far, so good. But to fully generate a working bean that can get and set the value of each of the color bean properties and notify the caller of its changes, we have to add event listeners to each of the set methods. There are two ways to do this. You could right-click the Bean Patterns node and choose Add > Multicast Event Source to add the `java.beans.propertyChangeListener` methods, then enter the rest of the source by hand.

An easier way is to generate all of the necessary code when you create the bean properties. First, let's get rid of all of the methods and fields we have created so far. You can do so by deleting the nodes from the Filesystems window or just by deleting the code in the Source Editor.

Next, right-click the Bean Patterns node and choose Add > Property. Enter `red` for the name, `int` for the type, and select the Bound checkbox. Now you can set the dialog to generate not just the field and methods for the property, but also the property change support code. Click OK to generate the following code in the Source Editor:

```
    private int red;

    private java.beans.PropertyChangeSupport propertyChangeSupport =  new java.beans.
PropertyChangeSupport(this);

    public void addPropertyChangeListener(java.beans.PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(java.beans.PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public int getRed() {
        return this.red;
    }

    public void setRed(int red) {
        int oldRed = this.red;
        this.red = red;
        propertyChangeSupport.firePropertyChange("red", new Integer(oldRed), new Integer(red));
    }
```

Then all you have to do is repeat the process for the `green` and `blue` properties and change the `ColorPreview` constructor to the following:

```
    public ColorPreview() {
        propertyChangeSupport = new java.beans.PropertyChangeSupport(this);
    }
```

And that's it! You've got a nice working bean ready to be used by the `ColorPicker` program.

## Working With Import Statements

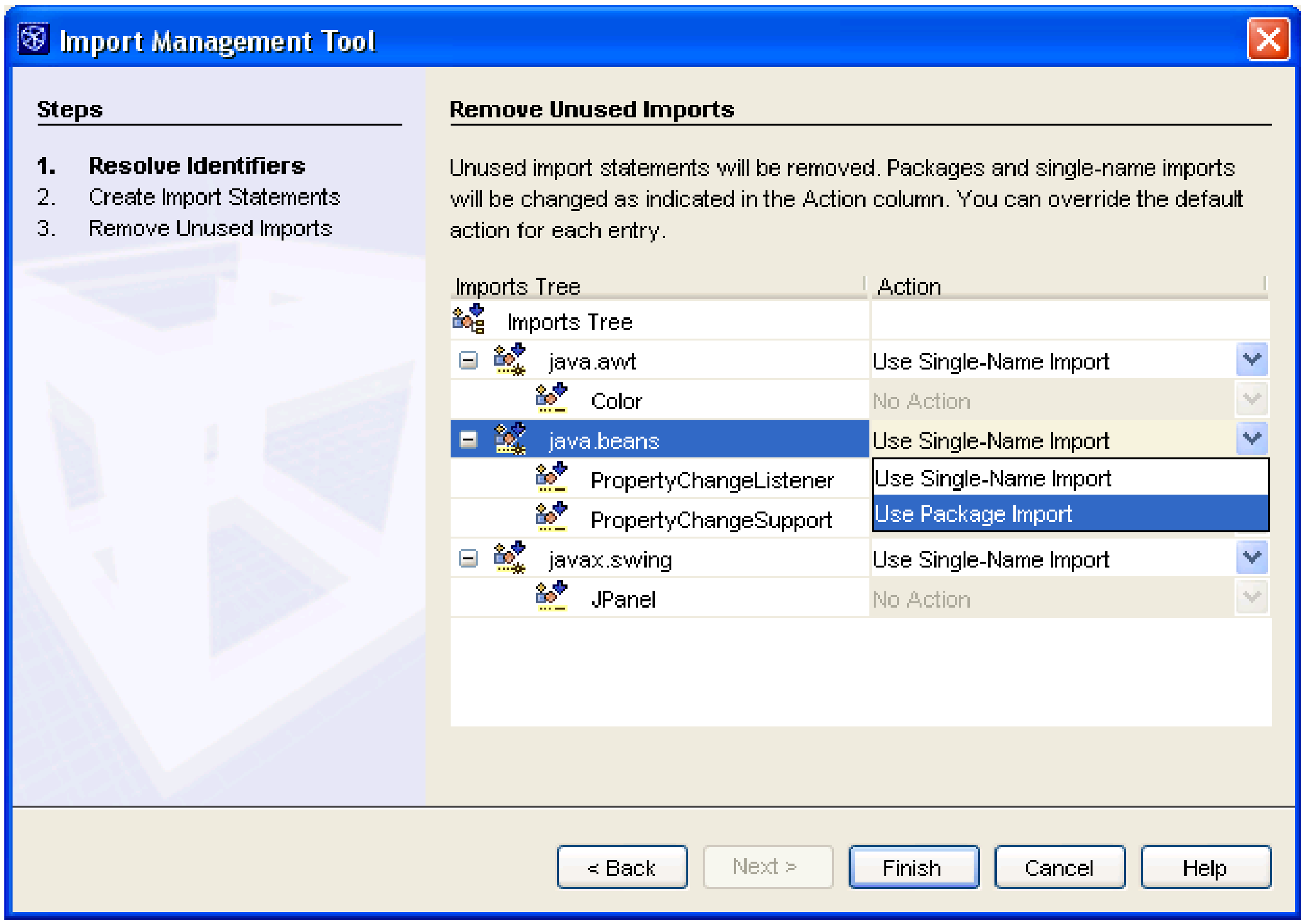
Whenever the IDE generates Java source code, it uses the fully qualified names for all the elements it creates. There are two tools that you can use to add import statements to your code and change between simple names and fully qualified names: the Fast Import command and the Import Management Tool.

To use the Fast Import command, place the insertion point on any class name and press Alt-Shift-I. In the following dialog box, specify whether to import the class or the entire package.

Unfortunately, the Fast Import command does not change all fully qualified names for the class to simple names. A more complete tool for handling import statements is the Import Management Tool (IMT). By default, the IMT changes all occurrences of fully qualified names into simple names and creates a single-name import statement for each.

Right-click anywhere in the `ColorPicker` file in the Source Editor and choose Tools > Import Management Tool. The first page of the IMT shows any unresolved identifiers in your file. These can occur when you incorrectly enter the class name or when you are referencing code that you do not have mounted in your project yet. You can enter a new package name to import for the classes, or import the classes as they are written.

At this point, you can click Finish immediately to run the IMT with its default settings. You can also click Next to further customize the tool's actions. For example, if you are importing several classes from a single package, you may want to import the entire package. You can do so on the Removed Unused Imports page of the wizard. Change the Action column for the package from Use Single-Name Import to Use Package Import.



**Remove Unused Imports**

Unused import statements will be removed. Packages and single-name imports will be changed as indicated in the Action column. You can override the default action for each entry.

| Imports Tree | Action |
|---|---|
| Imports Tree | |
| java.awt | Use Single-Name Import |
| Color | No Action |
| java.beans | Use Single-Name Import |
| PropertyChangeListener | Use Single-Name Import |
| PropertyChangeSupport | Use Package Import |
| javax.swing | Use Single-Name Import |
| JPanel | No Action |

## Search and Selection Tools

When you are dealing with a large group of files, the ability to quickly find, navigate to, and select certain strings or files is critical to your productivity. The following list gives you a quick overview of the search and selection tools that are available in the Source Editor:

| **Keyboard Shortcut** | **Description of Command** |
|---|---|
| Ctrl-F | Search for text in the currently selected file. The Source Editor jumps to the first occurrence of the string and highlights all matching strings. You can use F3 to jump to the next occurrence and Shift-F3 to jump to the previous. |
| Ctrl-H | Replace text in the currently selected file. |
| F3 | Find the next occurrence of the word you searched for. |

| | |
|---|---|
| Shift-F3 | Find the previous occurrence of the word you searched for. |
| Ctrl-F3 | Search for the next occurrence of the word that the insertion point is on. |
| Alt-Shift-H | Turn off search result highlighting. |
| Alt-Shift-O | Open the Fast Open dialog box, which lets you quickly open a file. Start typing a class name in the dialog box. As you type, all files that match the typed prefix are shown. The list of files is generated from the the project's mounted filesystems. |
| Alt-O | Go to source. This shortcut opens the file where the item at the insertion point is defined. |
| Alt-G | Go to declaration. Similar to the previous shortcut, this opens the file where the variable at the insertion point is declared. |
| Ctrl-G | Go to line. Enter any line number for the current file and press Enter to jump to that line. |
| Ctrl-F2 | Add a bookmark ( ) to the line of code that the insertion point is currently on. If the line already contains a bookmark, this command removes the bookmark. |
| F2 | Go to the next bookmark. |
| Alt-L | Go to the next location in the jump list for the currently selected file. The jump list is a history of all locations where you made modifications in the Editor. |
| Alt-K | Go to the previous location in the jump list for the currently selected file. |
| Alt-Shift-L | Go to the next jump list location in all files (not the currently selected file). |
| Alt-Shift-K | Go to the previous jump list location in all files. |

## Formatting Java Source Code

The IDE automatically formats your code as you write it. You can automatically reformat specific lines of code or entire files. The following table lists some common formatting commands.

| **Keyboard Shortcut** | **Description of Command** |
|---|---|
| Ctrl-Shift-F | Reformat the entire file or whatever text is selected in the Source Editor. |
| Ctrl-T | Shift the current line or selection one tab to the right. |
| Ctrl-D | Shift the current line or selection one tab to the left. |
| Ctrl-E | Remove the current line. |

| | |
|---|---|
| Ctrl-Shift-T | Comment out the current line or all selected lines with line comments ("//"). |
| Ctrl-Shift-D | Remove comments. This command only works for lines that begin with line comments ("//"). |

## Navigating Between Documents

The Source Editor makes it easy to manage large number of open documents at one time. The Source Editor displays a row of tabs for open documents. The tabs appear in the order in which you opened the documents. You can grab any tab and drag it along the row of tabs to move its position. Use the left and right buttons in the top-right corner to scroll through the row of tabs.

To switch between open files, do any of the following:

- Use the drop down list at the top-right of Source Editor. The drop down list displays all of your open files in alphabetical order.
- Press Alt-Left and Alt-Right to move one editor tab to the left or right.
- Press Ctrl-` to open the IDE window manager, which contains icons for each open document in the Source Editor as well as all open windows like the Filesystems window.

You can also:

- **Maximize the Source Editor**. Double-click any document tab or press Shift-Escape to hide all other IDE windows. If you have split the Source Editor, only the partition you maximize is displayed.
- **Clone a document.** Right-click the document in the Source Editor and choose Clone Document.
- **Split the Source Editor.** Grabbing any document tab and drag it to the left or bottom margin of the Source Editor. A red box shows you where the new Source Editor partition will reside once you drop the document. Any Source Editor partition can also be split any number of times.
- **Move documents between Source Editor partitions.** Grab the document tab and drag it to the row of tabs in the destination partition.

## Configuring the Editor

To configure Source Editor settings, open the Options window and expand Editing > Editor Settings. The Editor Settings node has subnodes for the editors used for each different file type. In this section, we will be looking at configuring the Java editor, but many of the settings are the same for all editors.

Here is a quick overview of some of the more common customizations to the Source Editor:

- **View or change abbreviations.** Open the property editor for the Abbreviations property and make any changes to the list.
- **View or change all keyboard shortcuts for the IDE.** Open the property editor for the Key Bindings property.
- **View or change all recorded macros.** Open the property editor for the Key Bindings property.
- **Turn off code completion.** Set the Auto Popup Completion Window property to False.

- **Set the font size and color for code.** Use the Font Size property to quickly change the font size for all Java code in the Source Editor. Open the property editor for Fonts and Colors to change the font and color of each type of Java code, like method names or strings.
- **Change the indentation used in your code.** You can switch between indentation engines by choosing a new engine from the Indentation Engine property. You can also configure each indentation engine by opening the property editor for the property.
- **Set how many spaces are inserted for each tab in your code.** Set the Tab Size property accordingly.
- **Turn off Javadoc for code completion.** Go to the Expert tab and set the Auto Popup Javadoc Window to False.

---

**Previous** - **TOC** - **Next**
**NetBeans IDE 3.5 version of this guide**

Top of page