

jProductivity LLC

*Productivity!* – The name speaks for itself

---

---

***Productivity! 2.0***  
*for Borland® JBuilder®*

***User Manual***

---

<http://www.jproductivity.com>

Copyright (c) 2000-2004 jProductivity L.L.C. All rights reserved.

JBuilder is registered trademark of Borland Software Corporation.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc in the United States and other countries.

Other brand and product names are trademarks or registered trademarks of their respective owners.

**Productivity! User Manual**

Copyright © 2000-2004 jProductivity L.L.C. <http://www.jproductivity.com>

## Contents

---

<b>PRODUCTIVITY! OVERVIEW .....</b>	<b>7</b>
<b>INSTALLATION INSTRUCTIONS.....</b>	<b>9</b>
<i>Productivity! Key Installation.....</i>	9
How to Obtain Key File .....	9
How to Install Key File .....	9
<i>Productivity! Help Installation.....</i>	9
<i>License Agreement Acceptance.....</i>	9
<i>Uninstalling Productivity! .....</i>	10
<b>COMPATIBILITY .....</b>	<b>11</b>
<b>PRODUCTIVITY! TOOLS .....</b>	<b>12</b>
<i>Common Insights Features .....</i>	16
Context Switching .....	16
Help Support .....	16
Code Generation Tools .....	17
<i>Class.Insight .....</i>	17
Class.Insight Actions .....	18
Showing Navigation Pane .....	18
Options Dependency .....	19
<i>Implement.Insight .....</i>	19
Code Changes Synchronization .....	20
Options Dependency .....	20
<i>Override.Insight and Constructor.Insight .....</i>	20
Code Changes Synchronization .....	21
Options Dependency .....	22
<i>Smart.Instantiate .....</i>	22
Showing Navigation Pane .....	23
Options Dependency .....	23
<i>GetSet.Creator .....</i>	23
<i>Introduce.Constructor – Pro! .....</i>	25
Options Dependency .....	26
<i>Delegate.Insight – Pro!.....</i>	26
Options Dependency .....	27
<i>Easy.JavaDoc and Easy.JavaDoc.Insight .....</i>	27
Easy.JavaDoc .....	28
Easy.JavaDoc.Insight.....	28
Options Dependency .....	29
Power Tools .....	29
<i>Rename Assistant - Pro! .....</i>	29
Options Dependency .....	30
<i>Assistant - Pro!.....</i>	30
Options Dependency .....	33
<i>Advanced To-Do's - Pro! .....</i>	33
Options Dependency .....	34
<i>Task List - Pro!.....</i>	34
Introduction to Tasks' Concept .....	35
Task List User Interface .....	35
Maintaining Tasks Using Task List .....	37
Reminders .....	38
Options Dependency .....	38

<i>Smart.Templates - Pro!</i> .....	39
Predefined Fields.....	41
Expressions.....	42
Smart.Templates.Insight.....	47
"On the Fly" Smart Templates .....	48
Options Dependency .....	49
<i>Smart.JavaDoc - Pro!</i> .....	50
Smart.JavaDoc User Interface .....	51
JavaDoc Editing Using Smart.JavaDoc .....	53
Smart.JavaDoc Toolbar.....	55
JavaDoc Errors Highlighting.....	59
Smart.JavaDoc Shortcuts.....	60
Editor Enhancements .....	62
<i>Smart.Clipboard - Pro!</i> .....	62
Paste Action .....	62
Copy/Cut Actions .....	62
Swap Action .....	62
Pop Paste Action .....	63
Clipboard.Insight .....	63
Options Dependency .....	63
<i>Smart.Selection - Pro!</i> .....	63
<i>Smart.Gutter - Pro!</i> .....	64
<i>Thumbnail Gutter - Pro!</i> .....	65
<i>Classes Highlight - Pro!</i> .....	66
<i>Advanced Text View Status - Pro!</i> .....	67
<i>Smart.Braces</i> .....	68
Options Dependency .....	68
<i>Matching.Code.Highlight - Pro!</i> .....	68
Options Dependency .....	69
<i>Smart.Braces.Highlight - Pro!</i> .....	69
Options Dependency .....	70
<i>Changes.Highlight - Pro!</i> .....	70
<i>Current Line Highlight - Pro!</i> .....	70
<i>Classes and Methods Separator - Pro!</i> .....	71
IDE Improvements .....	72
<i>Project View Synchronizer - Pro!</i> .....	72
Options Dependency .....	73
<i>Structure View Synchronizer - Pro!</i> .....	73
<i>Change.ReadOnly- Pro!</i> .....	73
Navigation Tools.....	75
<i>Browse.Insight</i> .....	75
Browse.Insight Actions .....	75
Options Dependency .....	76
<i>Browse.Members</i> .....	76
<i>Hyperlink.Navigate</i> .....	77
Options Dependency .....	77
<i>Search Results and References Highlight - Pro!</i> .....	77
<i>Persistent Bookmarks - Pro!</i> .....	78
Persistent.Bookmarks.Navigate.....	79
Manage Bookmarks Dialog .....	80
<i>View Navigator and Navigator.Insight - Pro!</i> .....	81
Information Tools .....	83
<i>Help.Insight</i> .....	83
Navigation Pane.....	83
Hyperlink.Help.....	84

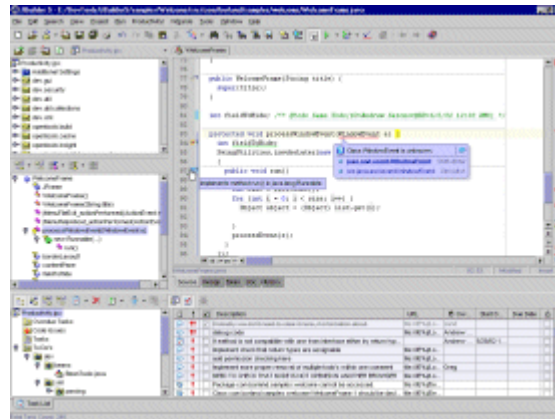
Integration with Other Insights .....	84
Options Dependency .....	85
<i>Hyperlink.Help</i> .....	85
Options Dependency .....	85
<i>Context.Insight</i> .....	85
Options Dependency .....	86
<b>PRODUCTIVITY! OPTIONS .....</b>	<b>87</b>
<i>Project Properties Dialog</i> .....	88
General Page.....	88
Code Style Page.....	89
JavaDoc Page .....	91
Cache Page .....	93
Assistant Page .....	95
Tools Page .....	97
<i>Editor Options Dialog</i> .....	98
Smart.Braces Options (Editor Options) .....	98
General Page.....	100
Usage Page .....	103
Delays Page .....	105
Tools Page – <i>Pro!</i> .....	106
Assistant Page – <i>Pro!</i> .....	108
Smart.JavaDoc Page – <i>Pro!</i> .....	109
Smart.Templates Page – <i>Pro!</i> .....	110
<i>Edit Template Dialog – Pro!</i> .....	112
General Page.....	112
Options Page .....	114
Fields Page.....	115
Shortcuts Page .....	117
<i>Edit Template Field Dialog – Pro!</i> .....	118
<i>IDE Options Dialog</i> .....	121
<b>PRODUCTIVITY! KEY BINDINGS .....</b>	<b>123</b>
<i>Key Bindings for CUA, Brief and Visual Studio keymaps</i> .....	124
<i>Key Bindings for Emacs, Macintosh and Macintosh Code Warrior keymaps</i> .....	126
<b>PRODUCTIVITY! TOOLS ICONS .....</b>	<b>129</b>
<b>KNOWN ISSUES AND LIMITATIONS .....</b>	<b>131</b>
<b>PRODUCTIVITY! FEEDBACK .....</b>	<b>133</b>

## Tables

---

Table 1 Productivity! Tools.....	12
Table 2 Productivity! Key Bindings for CUA, Brief and Visual Studio keymaps .....	124
Table 3 Productivity! Key Key Bindings for Emacs, Macintosh and Macintosh Code Warrior keymaps .....	126
Table 4 Productivity! Icons.....	129

# Productivity! Overview



Productivity! is a genuine and rich set of tools intended to greatly simplify routine coding and navigation operations. As a result, it allows significantly greater development productivity. All Productivity! tools are carefully designed and tuned to minimize efforts to invoke and use them so you can enjoy the friendly environment Productivity! offers.

With Productivity! tools:

- Be aware of any errors in your code and get assistance to fix them!
- Be always on schedule with help of Task List!
- Write bulletproof documentation for your code!
- Easily reuse your favorite code fragments!
- Write well-composed and easy maintainable code!
- Forget about typing your imports!
- Forget about annoying dialogs and Wizards while you are coding!
- Discover context and navigate through it!
- Use hyperlinks to surf and to get informed!
- Navigate freely through your classes, methods, fields and even editing points!
- Obtain quick help on classes and methods exactly where and when you need!
- Add super interfaces, change super classes in several simple steps!
- Override methods and constructors in a couple of clicks!
- Add access methods for you fields instantly!
- Use your own unique naming standards!

## Productivity! User Manual

- And finally, forget that you are using Productivity! - just enjoy your favorite IDE, interesting work and your superior performance!

Use Productivity! to add unleashed power to your JBuilder environment!



## Installation instructions

---

To install Productivity! you should unpack the archive you've downloaded and copy **productivity.jar** (**productivityPro.jar** for Professional Edition) to the *lib/ext* directory under your JBuilder installation.

**NOTE:** If you already have Productivity! installed in your system and you are going to install Productivity! Pro, please remove it since all functionality of Productivity! is included to Productivity! Pro. Also, you should remove previous versions of Productivity!/Productivity! Pro (if any).

### Productivity! Key Installation

---

Productivity! requires a key file, which enables the Productivity! functionality. In standard edition this file is named **productivity.key**, while in Professional Edition file name is **productivityPro.key**.

#### How to Obtain Key File

---

In some cases, the evaluation key file can be found in the downloaded archive. Otherwise, please visit <http://www.jproductivity.com> or contact <mailto:sales@jproductivity.com> to obtain an evaluation or commercial key.

#### How to Install Key File

---

The key file should be located in the same directory as used by JBuilder for storing its preferences and license. The location of this directory depends on the operating system installed on your computer.

Browse your HOME directory (you can find it using the *Home* button in the JBuilder *Open File dialog*). In the home directory you'll find the *.jbuilderX* (or *.jbuilder9*, *.jbuilder8*, *.jbuilder7*) depending on your version of JBuilder) subdirectory, where the key file should be placed.

Another way to find the location where the Productivity! key file should be placed is starting up JBuilder with Productivity! installed. If there is no key file, Productivity! will inform you of the fact with the appropriate message dialog; from this dialog, you can conclude about the location of the key file.

To install the key file, just copy the key file (**productivity.key** or **productivityPro.key**) to the location as specified above.

### Productivity! Help Installation

---

To install documentation for Productivity! please copy **productivity\_docs.jar** to the *doc* directory under your JBuilder installation.

### License Agreement Acceptance

---

After the first start of JBuilder with Productivity! Installed you'll be prompted to accept the Productivity! License Agreement. It should be accepted to allow running any Productivity! Tools. You can reject the Productivity! License Agreement and can accept it later using the Help | About Productivity! Dialog.

#### Productivity! User Manual

## Uninstalling Productivity!

---

To uninstall Productivity! please close JBuilder and remove copied jars and the key file.

## Compatibility

---

Productivity! 2.0 supports JBuilder version X only while Productivity! 1.x supports any JBuilder version from 4 up to 9. It doesn't impose big limitations related to JBuilder edition or host platform.

Please note, to run Productivity! Pro 1.X under JBuilder4 the *xerces.jar* should be installed in the system and path to it should be stated in the *classpath*.

Productivity! Pro edition is known to be not compatible or providing functionality that overlaps with the following JBuilder Open Tools:

- Syntax Checker (by Steven Lee);
- Extended Highlighting (by Volker Malzahn);
- Selection Margin 2.01 (by Karl Tauber);
- Number Line (by Gillmer Derge);
- Highlight Matching Parenthesis, Brace or Bracket (by Gillmer Derge);
- Java Node Icon Tip (by Keith Wood);
- RefactorIt (by Aqris Software AS);
- Text Drag Drop (by Karl Tauber);
- ChangeReadOnly (by Luke Hutterman);
- Tag Read Only (by Fabrizio Giustina);
- Structure Synchronizer (by Brian Sayatovic);
- Synchronize ProjectView and Broser selection (by Torsten Welches);
- Where Am I? By David Pierron;
- JbPropStructure (by Angus Chan);
- Clipboard Manager (by Jacob Roberson);

## Productivity! Tools

---

Productivity! Offers a powerful set of tools intended to reduce routine coding tasks. These tools are carefully designed to allow solving such tasks with minimum efforts and in minimal time.

The following tools are available after installing Productivity!.

**NOTE:** The exact set of tools included depends on edition – the Pro! mark highlights tools available in Productivity! Professional Edition only.

All tools offered by Productivity! belong to the following groups:

- Code Generation Tools
- Power Tools
- Editor Enhancements
- IDE improvements
- Navigation Tools
- Information Tools

Please refer the table below to find more about content of these groups. Also, there you can find short description of every tool included into Productivity!

**Table 1 Productivity! Pro Tools**

Tool	Description
<b>Code Generation Tools</b>	
<i>Pro!</i> <a href="#">Delegate.Insight</a>	<a href="#">Delegate.Insight</a> provides an easy way to generate methods, which implementations are delegated to another object (delegate).
<i>Pro!</i> <a href="#">Introduce.Constructor</a>	<a href="#">Introduce.Constructor</a> allows easy generation of constructors intended to initialize appropriate fields of the class.
<a href="#">Class.Insight</a>	<a href="#">Class.Insight</a> allows quick finding Java classes with short names matching the word at the cursor position, and inserting the class name found into the cursor position as well as inserting import statement.
<a href="#">Implement.Insight</a>	<a href="#">Implement.Insight</a> allows quick finding Java classes with short names matching the word at the cursor position and using them either as a super interface or as a super class.
<a href="#">Override.Insight</a>	<a href="#">Override.Insight</a> allows quick finding methods to override with names matching the word at the cursor position or a typed word, and overriding them into the class at the cursor

	position.
Constructor.Insight	Constructor.Insight allows quick overriding class constructors.
Easy.JavaDoc	Easy.JavaDoc allows easy and convenient generating templates for JavaDoc comments for a particular method or class.
Smart.Instantiate	Smart.Instantiate is an additional Class.Insight functionality that allows adding instantiation of a particular class or interface.
GetSet.Creator	GetSet.Creator is a tool that allows easy creation of accessors and/or mutators for selected fields of a class.
<b>Power Tools</b>	
<i>Pro!</i> Rename Assistant	This tool simplifies identifiers renaming by introducing "in-place" rename approach.
<i>Pro!</i> Assistants	The set of assistants those show information about the particular issue and/or list of possible actions to complete.
<i>Pro!</i> Task List	The Task List is a tool that allows viewing and managing the list of tasks, code issues and to-do's.
<i>Pro!</i> Advanced To-Do's	The Advanced To Do's tool is further expansion of the To Do comments concept by treating them as tasks with set of attributes like priority, status, owner etc.
<i>Pro!</i> Smart.Templates	The Smart.Templates is advanced template engine that provides set of sophisticated features, like linked fields, calculated fields; expression and functions supports. It introduces advanced code templates those can be easily adapted to particular coding style coding style.
<i>Pro!</i> Smart.JavaDoc	The Smart.JavaDoc tool is additional viewer for Java file node that offers JavaDoc authoring in mode that is very close to WYSIWYG one.
<b>Editor Enhancements</b>	
Smart.Braces	The Smart.Braces is a tool that allows easy creation of closing braces while you are typing.
<i>Pro!</i> Code Folding Enhancements	Ability to fold Java Doc comments and collapse all of them (by using special button on the view toolbar.
<i>Pro!</i> Classes Highlight	Allows highlighting classes used in the code.
<i>Pro!</i> Thumbnail Gutter	The Thumbnail Gutter represents additional gutter placed near the vertical scrollbar of editor pane and allows showing gutter marks for the whole source file as well as navigate to them.
<i>Pro!</i> Smart.Clipboard	The Smart.Clipboard represents several tools used for provide more efficient clipboard operations. These tools

	include clipboard management, swapping context of clipboard with current selection, inserting pasted Java code with correct indent along with required import statements etc.
<b>Pro!</b> Smart.Gutter	The <b>Smart.Gutter</b> is additional gutter placed near standard JBuilder editor gutter and is used for showing various hints concerning corresponding code in editor by arranging appropriate gutter marks.
<b>Pro!</b> Smart.Braces.Highlight	The <b>Smart.Braces.Highlight</b> tool provides matching braces highlight and navigation operations as well as showing code fragment that corresponds to appropriate brace.
<b>Pro!</b> Matching.Code.Highlight	The <b>Matching.Code.Highlight</b> tool performs highlighting of code matching to one at caret position as well as displaying appropriate code fragment in the popup window.
<b>Pro!</b> Changes Highlight	The <b>Changes Highlight</b> Highlights changed lines if Java source on the gutter.
<b>Pro!</b> Methods and Classes Separator	The <b>Methods and Classes Separator</b> tool visually separates classes and methods from each other by painting horizontal line at the top of declaration.
<b>Pro!</b> Current Line Highlight	The <b>Current Line Highlight</b> tool highlights the line under cursor in the current editor with appropriate background color.
<b>Pro!</b> Smart.Selection	The <b>Smart.Selection</b> tool offers sophisticated code selection functionality that is based on n structure of Java program. It allows expanding/narrowing selection incrementally using appropriate code elements as well as quickly selection of whole statement, code block, method or class.
<b>Pro!</b> Advanced Text View Status Bar	The <b>Advanced Text View Status Bar</b> represents a replacement of standard component that allows viewing of class and method for current caret position, caret offset, and lines count.
<b>IDE Improvements</b>	
<b>Pro!</b> Project View Synchronizer	This tool provides functionality for synchronizing of currently active file with the corresponding node in the JBuilder Project View.
<b>Pro!</b> Java Structure Synchronizer	The <b>Java Structure Synchronizer</b> allows synchronizing the Java Structure View with current caret position in the editor.
<b>Pro!</b> Change.ReadOnly	The <b>Change.ReadOnly</b> allows easy viewing and managing the read-only status for file nodes.
<b>Navigation Tools</b>	
<b>Pro!</b> Persistent.Bookmarks	The <b>Persistent.Bookmarks</b> tool offers advanced bookmarks functionality. These bookmarks are persistent between sessions of JBuilder, are associated not only with editor, but also with file and JBuilder project.

<i>Pro!</i> View Navigator	The <a href="#">View Navigator</a> tool allows quick navigation by source elements (classes, methods, fields), issues/errors, editing points or search results.
<i>Pro!</i> Navigator.Insight	The <a href="#">Navigator.Insight</a> is specialized insight used for quick controlling of <a href="#">View Navigator</a> .
Browse.Insight	The <a href="#">Browse.Insight</a> tool allows quick finding Java classes with short names matching the word at the cursor position and browsing them or the appropriate help topics.
Browse.Members	The <a href="#">Browse.Members</a> tool allows quick finding members belonging to the current discovered context and browsing them.
Hyperlink.Navigate	<a href="#">Hyperlink.Navigate</a> is a tool that allows easy and convenient navigation through symbols definitions basing on the concept of hyperlinks.
<i>Pro!</i> Search Results and References Highlight	This tool is intended to highlight in the editor various things found during search or find references operations.
<i>Pro!</i> Local References Highlight	This tool allows finding local references of the symbol under caret and highlighting them.
<b>Information Tools</b>	
Help.Insight	<a href="#">Help.Insight</a> allows easy viewing help topics, if any, for the identifier at the cursor position. Also, it provides quick help for items shown in JBuilder built-in Member Insight and Productivity! insights.
Hyperlink.Help	<a href="#">Hyperlink.Help</a> allows easy and convenient viewing help topics for particular symbols.
Context.Insight	<a href="#">Context.Insight</a> is a tool that allows you to check context of the current cursor position. <a href="#">Context.Insight</a> collects information about all classes and methods and shows it using the insight popup window.

## Common Insights Features

---

Most of Productivity! Insights share the following approaches.

### Context Switching

---

During invocation, any context dependent Insight analyses context structure and selects the target for modification: the deepest class or method found for the cursor position.

The Context label shows the full-qualified name of this class or method. If there are several classes or methods found in the context path you can choose a different class as a target. Use *Switch Context up* to and *Switch Context down* to buttons or keyboard shortcuts **Alt+Up** or **Alt+Down**, respectively, to select a class or methods as the target; the Context label will reflect the changes. This functionality is useful when cursor is placed within an inner class while you need to execute appropriate actions to the outer one.

### Help Support

---

To view help press the appropriate key mapped to the help action in the current keymap (typically, this is **F1**).

If an Insight shows the list of members and there is a member (either class, method or field) selected in the list, the Help Viewer will show the appropriate documentation page for this member (if any). If the members list is empty or there is no member selected, the help on the Insight will be shown. You can use the *Help* button in the Navigation Pane to invoke help on the Insight directly.

[Help.Insight](#) is an Alternative way of getting help on a selected member. To allow [Help.Insight](#) invocation when using the Insight you should turn on the *Editor Options | Productivity! | Usage | Integrate Help.Insight with Productivity! Insights* checkbox. With this option turned on, just select a member and wait until [Help.Insight](#) popup will show the appropriate JavaDoc help page (if any).

You can also force [Help.Insight](#) invocation using the shortcut **Shift+F1** (CUA). You can specify [Help.Insight](#) invocation delay using the *Editor Options | Productivity! | Delays | Help.Insight Invocation Delay* slider.



## Code Generation Tools

---

Productivity! offers powerful set of code generation tools intended to simplify routing but very common operations like:

- Inserting appropriate import statements for class (using short class name) – the [Class.Insight](#) tool;
- Implementing interface or extending class – the [Implement.Insight](#) tool;
- Overriding methods – the [Override.Insight](#) tool;
- Creation of constructor with the same signature as one defined in super class – the [Constructor.Insight](#) tool;
- Instantiation of variable - the [Smart.Instantiate](#) tool;
- Creating of getter/setter methods – the [GetSet.Creator](#) tool;
- Creating of constructor used to initialize set of class fields – the [Introduce.Constructor](#) tool;
- Creating of proxy delegate methods those actually calls methods of class member – the [Delegate.Insight](#) tool
- Creation of default JavaDoc comment – the [Easy.JavaDoc](#) tool;
- Import statements optimization – the [Imports.Beautify](#) tool.

All these tools provide very simple and intuitive interface and allow you to get required task complete using minimal amount of actions needed.

### Class.Insight

---

[Class.Insight](#) - Forget about typing your import statements!

[Class.Insight](#) allows quick finding Java classes with short names matching the word at the cursor position, and inserting there the class name found and its import statement. To choose a class from several possible variants, it employs a popup window similar to other JavaInsight popups (MemberInsight, ParameterInsight etc.). You don't need to type import statements manually - just use [Class.Insight](#) to find and insert the required class and let it make all other job for you!

The [Class.Insight](#) backend caches all the required information about classes containing in project, JDK and project required libraries to speedup usage. The cache build is initiated only on the first [Class.Insight](#) invocation so it doesn't affect JBuilder startup and a project opening time.

The [Class.Insight](#) saves the cache in the project directory while project closing and loads the cache from disk when the project is opened next time. The cache file is named <Project Name>.cache and it can be easily removed when unneeded us Productivity! will automatically recreate it before the next [Class.Insight](#) use.

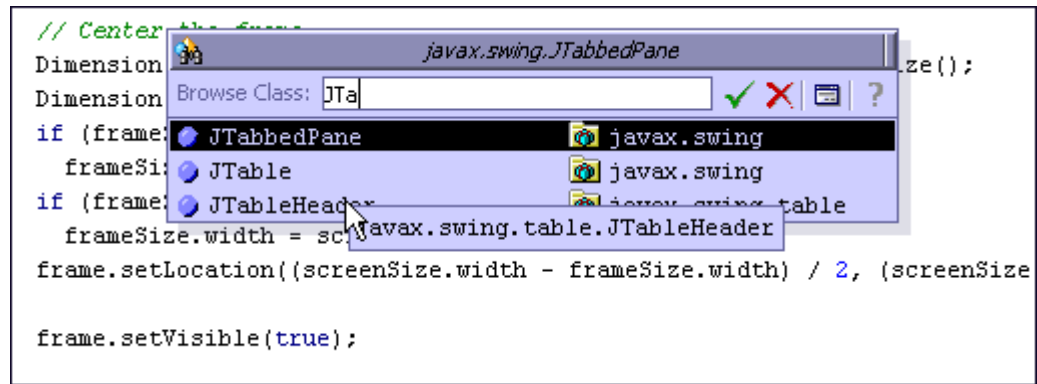


Figure 1 Class.Insight Popup Window

When editing a file, place the cursor over the word you want to expand as a class name (or at a blank space) and press **Ctrl+Alt+Space** (**Ctrl+Alt+H**) (CUA) to invoke **Class.Insight**. The **Class.Insight** popup will be shown with the list of classes matching the word at the cursor position. You can select a class navigating through the list with the help of the usual keyboard. An Alternative way to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

### Class.Insight Actions

---

On selecting a class, you may choose from several options with the help of the following shortcuts:

**Enter:** - **Class.Insight** replaces the word at the cursor position with a class name and adds the appropriate import statement;

**Ctrl+Enter:** - **Class.Insight** replaces the word at the cursor position with a full-qualified class name;

**Alt+Enter:** - **Class.Insight** switches between importing a particular class and the whole package.

**Shift+Enter:** - **Class.Insight** produces a code for instantiation of the selected class variable.

If there are no matches found, *Java.Insight - Select Class dialog* is shown. Select a class in this dialog and press OK. The **Class.Insight** will replace (or just insert) the word at the cursor position with a selected class name adding the appropriate import statement.

### Showing Navigation Pane

---

You can switch **Class.Insight** popup to show the Navigation Pane by turning off the *Editor Options / Productivity! / Usage / Show Class.Insight popup as list* checkbox. With this option turned off, **Class.Insight** popup will be shown with the Navigation Pane, that allows using **Class.Insight** popup even if there is no word at the cursor position or if there are no matching classes found. To find matches, type a word in the *Use Class* edit box and **Class.Insight** will dynamically rearrange the classes' list to show the matching ones.

## Options Dependency

---

Please note that the set of classes shown in the [Class.Insight](#) list depends on Packages Exclusion settings on the *Project Properties | Productivity! | General* property page.

Import statements are generated basing on Imports Generation settings on the *Editor Options | Productivity! | General* property page. There you can also customize other [Class.Insight](#) options, such as *Search Options*, *Sort Classes By*, *Autocomplete* and *Productivity! Insights Usage*

## Implement.Insight

---

[Implement.Insight](#) allows quick finding of Java classes with short names matching the word at the cursor position and using them either as a super interface or super class for the class at the cursor position.

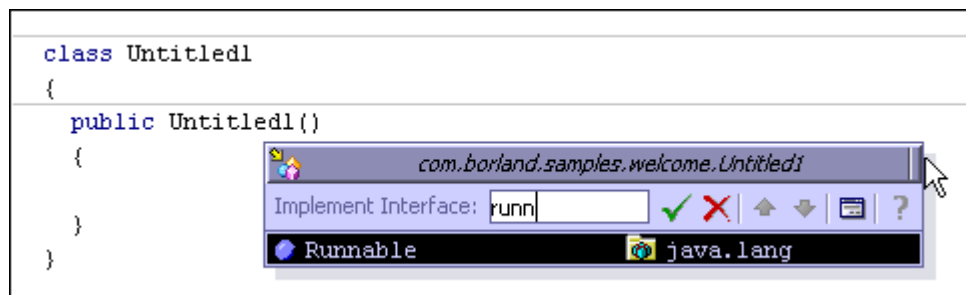


Figure 2 [Implement.Insight](#) Popup Window

When editing a file, place the cursor within the bounds of the class you want to add a super interface or set a super class to, and press **Ctrl+Alt+I** (CUA) to invoke [Implement.Insight](#). The [Implement.Insight](#) popup will be shown with the list of classes matching the word at the cursor position. The list may be empty if there are no matching classes though. To find matches, type the word in the [Implement Interface](#) edit box and [Implement.Insight](#) will dynamically rearrange the classes' list to show the matching ones.

You can select a class navigating through the list with the help of the usual keyboard. An Alternative way to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

Press the **Enter** key when you find the required class and [Implement.Insight](#) will add this class to the list of super interfaces or set it as the super class for the target one. [Implement.Insight](#) will also write all the methods defined in the interface or all the abstract methods defined in the class (if you have selected an interface and a class, respectively).

If you have selected a class (not an interface) and the target one already has a super class you will be prompted to confirm modifications.

Also, there is a possibility of invoking the built-in [Implement Interface Wizard](#). You can use the appropriate button in the top left corner of the popup.

```

public class Untitled1 implements Runnable
{
    public Untitled1()
    {
    }

    public void run()
    {
        /**@todo: Implement this run() method*/
        throw new UnsupportedOperationException("Method run() not yet implemented.");
    }
}

```

Figure 3 Code Generated by `Implement.Insight`

The figure above illustrates code generated by `Implement.Insight`.

### Code Changes Synchronization

---

`Implement.Insight` analyses changes in all dependant source files and correctly reflects them during generation of abstract methods implementations. But for most of the cases you need to compile all dependant classes before invocation of `Implement.Insight`. If the required class is not compiled yet or the required methods are not found in the compiled class, these errors will be shown in the Status View.

### Options Dependency

---

Please note that the set of classes shown in `Implement.Insight` list depends on *Packages Exclusion* settings on the *Project Properties | Productivity! | General* property page. Also there you can customize *Code Generation Options*, which allow you to adjust the code style for the generated methods code.

Import statements will be generated basing on *Imports Generation* settings on the *Editor Options | Productivity! | General* property page. There you can also customize other `Implement.Insight` options, such as *Search Options* and *Sort Classes By*.

### Override.Insight and Constructor.Insight

---

`Override.Insight` allows quick finding of methods and constructors to override with names matching a word at the cursor position or a typed word and overriding them in the class at the cursor position.

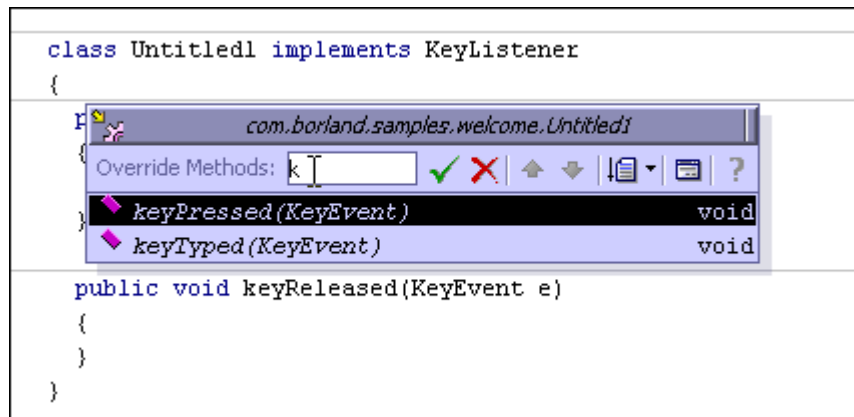


Figure 4 Override.Insight Popup Window

When editing a file, place the cursor within the bounds of the class you want to override methods for, and press **Ctrl+M** (CUA) to invoke **Override.Insight**. The **Override.Insight** popup will be shown with the list of methods those match the word at the cursor position. The list may be empty if there are no matching methods though. To find matches, type a word in the **Override Methods** edit box and **Override.Insight** will dynamically rearrange the methods' list to show the matching ones. You can also leave the **Override Methods** edit box blank to view all the methods to override. **Override.Insight** highlights the methods with names exactly matching the typed word with bold font and the abstract methods with italic font.

In addition to the methods inherited from the super class, **Override.Insight** shows the methods defined in the interfaces but not implemented directly by the target class.

You can select a method, either one or any, navigating through the list with the help of the usual keyboard. An Alternative way to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

Press the **Enter** key when you select the required methods and **Override.Insight** will override them and add calls to the appropriate methods of the super class, if needed.

You can call **Override.Insight** with constructors only using the shortcut **Ctrl+Shift+M** (CUA).

Also, there is a possibility of invoking the built-in **Override Methods Wizard**. You can use the appropriate button in the left top corner of the popup to invoke it.

### Code Changes Synchronization

**Override.Insight** analyses changes in all dependant source files and correctly reflects them in the methods list. But in most cases you need to compile all dependant classes before invoking **Override.Insight**. If the required class is not compiled yet or the required methods are not found in the compiled class, these errors will be shown in the **Status View**.

## Options Dependency

---

Using the *Project Properties | Productivity! | General* property page you can customize *Code Generation Options*, which allow you to adjust the code style for the generated methods code.

Import statements are generated basing on *Imports Generation* settings on the *Editor Options | Productivity! | General* property page. There you also can customize other *Override.Insight* options, such as *Search Options* and *Sort Classes By*.

## Smart.Instantiate

---

*Smart.Instantiate* is an additional functionality of *Class.Insight* that allows adding instantiation of a particular class by invoking *Class.Insight*, selecting the class and pressing **Shift+Enter**.

*Smart.Instantiate* recognizes the need to define a variable or just to create a new object. For example, when you type `List fList = new List(100);` and use *Smart.Instantiate* to create an `ArrayList` instance, *Class.Insight* replaces only the appropriate class name and preserves the variable definition and constructor parameters. You will get the following `List fList = new ArrayList(100);`

The same behavior is exhibited when using *Smart.Instantiate* to create a new instance and as a parameter to a method call. In other cases, *Smart.Instantiate* inserts definition and initialization of the variable with a new instance of the selected class.

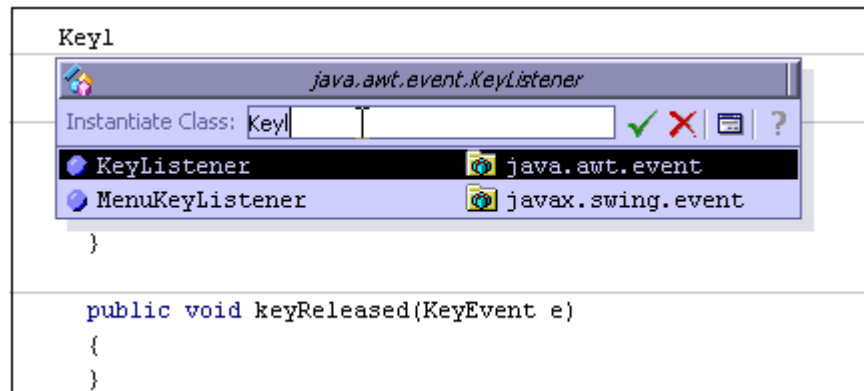


Figure 5 *Smart.Instantiate* Popup Window

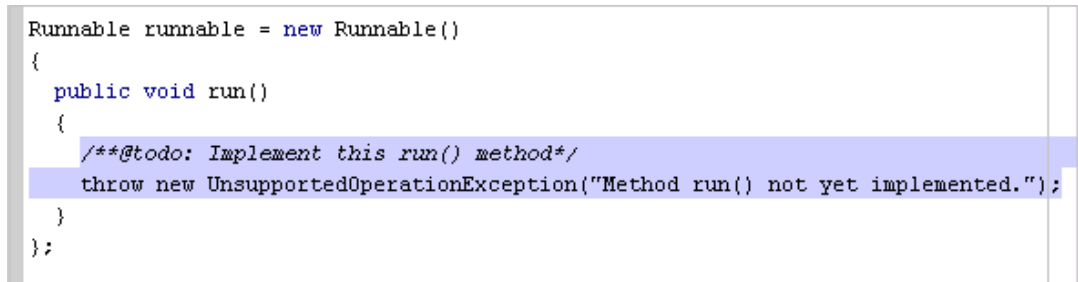
When an interface is selected to be instantiated, *Smart.Instantiate* automatically inserts implementation of the interface as an anonymous inner class. You can control this behavior using the *Project Properties | Class.Insight | General* property page.

An Alternative way to get *Smart.Instantiate* executed is using the shortcut **Alt+I** (CUA) that invokes a particular *Smart.Instantiate* popup. This popup is similar to the *Class.Insight* one but it doesn't require holding the **Shift** key to activate *Smart.Instantiate* - you just need to select a class and press the **Enter** key to instantiate it.

A couple of samples of illustrating what *Smart.Instantiate* can do for you!



Figure 6 Code Before Invocation of Smart.Instantiate



```
Runnable runnable = new Runnable()
{
    public void run()
    {
        /**@todo: Implement this run() method*/
        throw new UnsupportedOperationException("Method run() not yet implemented.");
    }
};
```

Figure 7 Code After Invocation of Smart.Instantiate

### Showing Navigation Pane

---

You can switch the [Smart.Instantiate](#) popup to show the Navigation Pane by turning off the *Editor Options | Productivity! | Usage | Show Class.Insight popup as list* checkbox. With this option turned off, [Smart.Instantiate](#) popup will be shown with the Navigation Pane, that allow to use [Smart.Instantiate](#) even if there is no word at the cursor position or if there are no classes matching it. To find matches, type a word in the *Instantiate Class* edit box and [Smart.Instantiate](#) will dynamically rearrange the classes' list to show the matching ones.

### Options Dependency

---

Please note that the set of classes shown in the [Smart.Instantiate](#) list depends on Packages Exclusion settings on the *Project Properties | Productivity! | General* property page.

Import statements are generated basing on Imports Generation settings on the *Editor Options | Productivity! | General* property page. There you can also customize other options of [Smart.Instantiate](#), such as *Search Options*, *Sort Classes By*, *Autocomplete* and *Productivity! Insights Usage*.

Using the *Project Properties | Productivity! | General* property page you can customize *Code Generation Options*, which allow you to adjust the code style for the generated methods code.

### GetSet.Creator

---

[GetSet.Creator](#) is a tool that allows easy creation of accessors and/or mutators for selected fields of a class.

When editing a file, press **Alt+Shift+A** (CUA) to invoke [GetSet.Creator](#) Insight. The [GetSet.Creator](#) popup will be shown with the list of fields matching a word at the cursor position. The list may be empty if there are no matching fields though. To find matches, type a word in the Fields edit box and [GetSet.Creator](#) will dynamically rearrange the list of fields to show the matching ones. You can also leave the Fields edit box blank to view all fields.

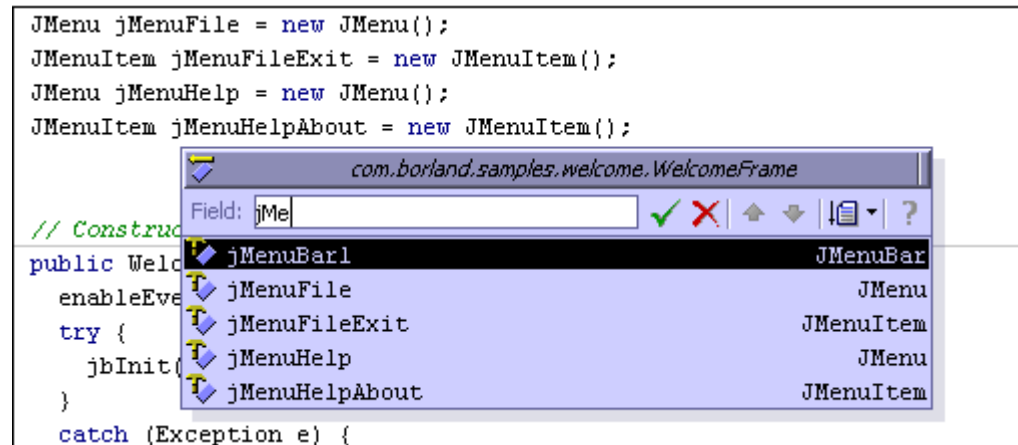


Figure 8 GetSet.Creator Popup Window

[GetSet.Creator](#) highlights the fields with names exactly matching the typed word using bold font.

[GetSet.Creator](#) analyses all the fields and all the methods those may be considered as accessor or mutator ones and removes certain fields from the list if appropriate methods are already exist.

You can select a field, either one or any, navigating through the list with the help of the usual keyboard. An Alternative way to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

Press the Enter key when you select the required field(s) and [GetSet.Creator](#) will generate applicable accessors and (or) mutators to it (you can select all items in the list using the **Ctrl+A** shortcut). When generating a method, [GetSet.Creator](#) analyses the current class as well as all its super classes and super interfaces, so it can call the appropriate method of the super class or skip particular method generation in case of any contradictions.

There is an ability to invoke [GetSet.Creator](#) in the mode that allow generating either accessors or mutators methods only using the **Alt+Shift+G** or **Alt+Shift+S** (CUA) shortcuts, respectively.

```

public JMenuBar getJMenuBar1() {
    return jMenuBar1;
}

public void setJMenuBar1(JMenuBar jMenuBar1Value) {
    jMenuBar1 = jMenuBar1Value;
}

```

Figure 9 Code Generated by GetSet.Creator

In general, [GetSet.Creator](#) uses Java Beans convention for naming the accessor and mutator methods. But if your code style assumes using prefixes or (and) suffixes for fields naming, [GetSet.Creator](#) allows you to use them without distortion of method



names - you just need to specify correct prefixes and suffixes in the *Project Properties* / *Productivity!* / *Code Style* / *Fields Naming* options group. For example, if you specify prefix `m_` and name your field as `m_count`, *GetSet.Creator* generates methods as `getCount()` and `setCount(...)`.

*GetSet.Creator* can generate JavaDoc comments during methods generation. To control this, please use the options on *the Project Properties* / *Productivity!* / *JavaDoc* property page.

### Introduce.Constructor – Pro!

*Introduce.Constructor* is a tool that allows easy generation of constructors those are intended to initialize selected class' fields. It can be invoked using **Alt+Shift+C** (CUA) shortcut. The *Introduce.Constructor* popup will be shown with the list of fields matching a word at the cursor position. The list may be empty if there are no matching fields though. To find matches, type a word in the Fields edit box and *Introduce.Constructor* will dynamically rearrange the list of fields to show the matching ones. The *Fields* edit box can be blank that allows viewing all fields.



Figure 10 Introduce.Constructor Popup Window

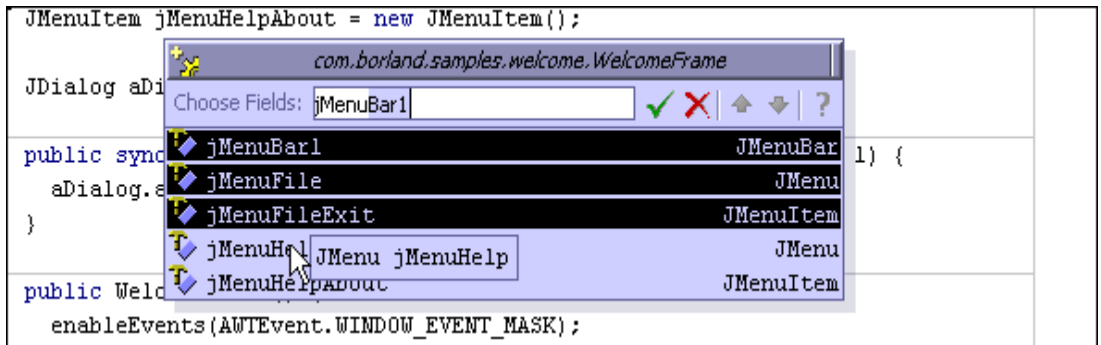


Figure 11 Selection of Fields Should be Initialized

One or more fields can be selected using the fields' list.

**NOTE:** The order of fields' selection exactly defines the definition order of constructor parameters and, in turn, the order of fields' initialization.

As soon as fields to initialize are selected pressing the **Enter** key closes [Introduce.Constructor](#) popup and introduces constructor that initializes all selected fields.

```
public WelcomeFrame(JMenuBar jMenuBar1, JMenu jMenuFile, JMenuItem jMenuItemExit)
{
    this.jMenuBar1 = jMenuBar1;
    this.jMenuFile = jMenuFile;
    this.jMenuItemExit = jMenuItemExit;
}
```

Figure 12 Code Generated by Introduce.Constructor

Please note that if constructor with the same signature is already exists, [Introduce.Constructor](#) popup will not be closed and appropriate error message will be displayed in the Status Bar.

**Options Dependency**

The *Project Properties | Productivity! | Code Style* property page allows customizing code style and placement for the generated constructors. Settings for Java Doc comments those can be optionally generated during constructors' introducing can be customized using the *Project Properties | Productivity! | Java Doc* property page.

**Delegate.Insight – Pro!**

The [Delegate.Insight](#) tool provides an easy way to generate methods those actual implementations are delegated to another object (delegate). To invoke [Delegate.Insight](#) please use **Alt+Shift+M** (CUA) shortcut.

After invocation, the [Delegate.Insight](#) popup is shown with the list of members (fields and methods) matching a word at the cursor position. The list may be empty if there are no matching or suitable members though. To find matches, type a word in the *Choose Delegate* edit box and [Delegate.Insight](#) will dynamically rearrange the list of members to show the matching ones. The *Choose Delegate* edit box can be blank that allows viewing of all members.

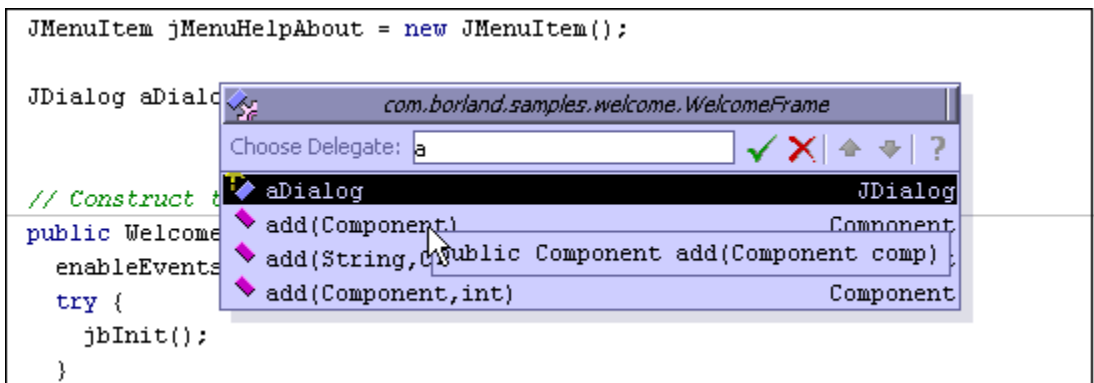


Figure 13 Delegate.Insight Popup Window – Selecting a Delegate

As soon as a member to become the delegate is selected (multiple selection is not supported by this tool) and press the **Enter** key, the [Delegate.Insight](#) popup shows the

list of methods belonging to the delegate and suitable to be introduced in the target class.

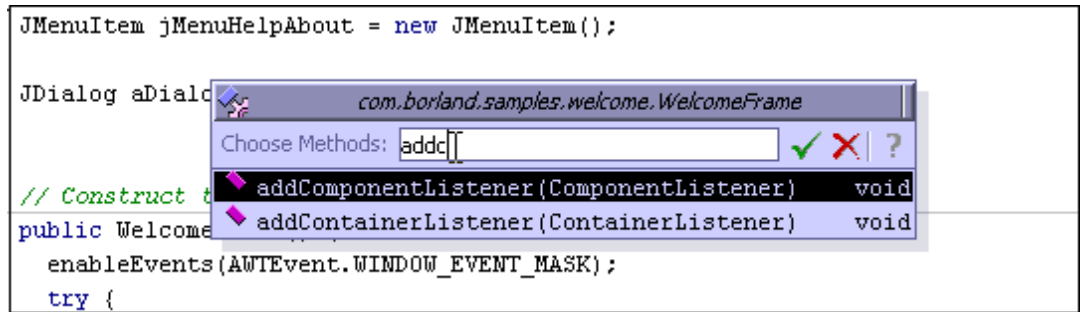


Figure 14 Delegate.Insight Popup Window – Selecting Methods

To find matches please type a word in the *Choose Methods* edit box and *Delegate.Insight* will dynamically rearrange the list of methods to show the matching ones. The *Choose Method* edit box can be blank that allows viewing all methods.

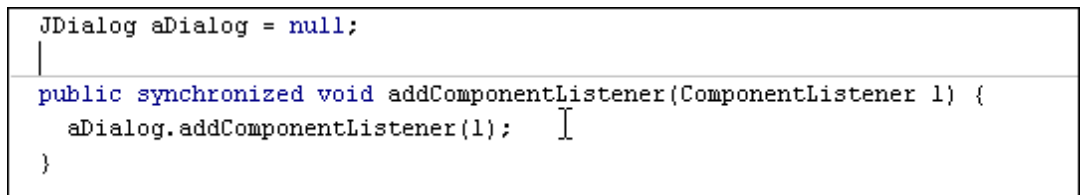


Figure 15 Code Generated by Delegate.Insight

*Delegate.Insight* correctly recognizes delegate modifiers and generates static methods if required (preserving other modifiers as well). It utilizes all major delegate patterns and allows using the following entities as delegates:

- Field declared either in currently edited class or in any of its parents;
- Execution result of method without parameters;
- Execution result of method with parameters. In this case *Delegate.Insight* generates methods, which have merged list of parameters - one part is needed to obtain a delegate and another one to be passed to delegates' methods call.

### Options Dependency

---

The *Project Properties | Productivity! | Code Style* property page allows customizing code style and placement for the generated methods. Settings for Java Doc comments those can be optionally generated during methods' generation can be customized using the *Project Properties | Productivity! | Java Doc* property page.

### Easy.JavaDoc and Easy.JavaDoc.Insight

---

*Easy.JavaDoc* is a tool that allows easy and convenient generating of templates for JavaDoc comments on particular methods or classes (except for the anonymous ones).

## Easy.JavaDoc

To invoke Easy.JavaDoc, place the cursor within a method or class for which you want to generate JavaDoc and press **Ctrl+D** (CUA). JavaDoc comment will be automatically inserted just before the method. Since the method comments contain tags for all declared fields, exceptions can be thrown by the method. That's really easy!

## Easy.JavaDoc.Insight

[Easy.JavaDoc.Insight](#) allows choosing of several methods or classes for JavaDoc generation.

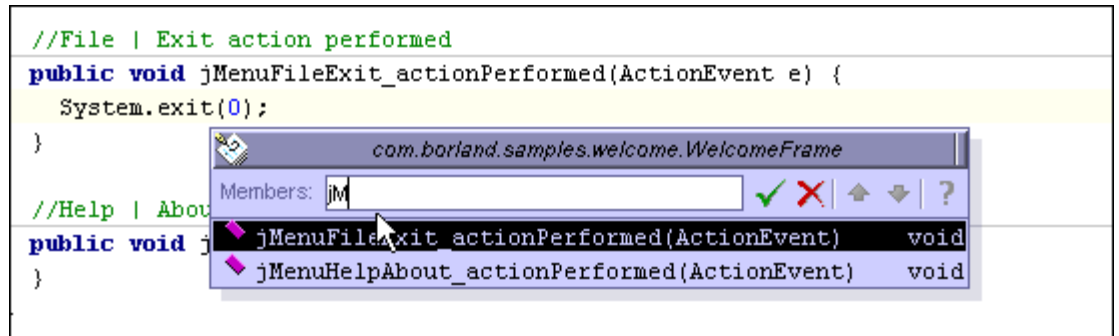


Figure 16 Easy.JavaDoc.Insight Popup Window

Press **Ctrl+Shift+D** (CUA) to invoke Easy.JavaDoc.Insight. The [Easy.JavaDoc.Insight](#) popup will be shown with the list of members (methods and/or classes) matching the word typed in the *Members* edit box. The list may be empty if there are no matching members though. To find matches, type a word in the *Members* edit box and [Easy.JavaDoc.Insight](#) will dynamically rearrange the members' list to show the matching ones. If you leave the *Members* edit box blank, all members within the current context are shown.

**NOTE:** Unlike other Insights, [Easy.JavaDoc.Insight](#) doesn't merely employ the word at the cursor position; it rather uses the name for the method or class at the cursor position. This approach allows easy generation of JavaDoc comments for the method or class at the cursor position.

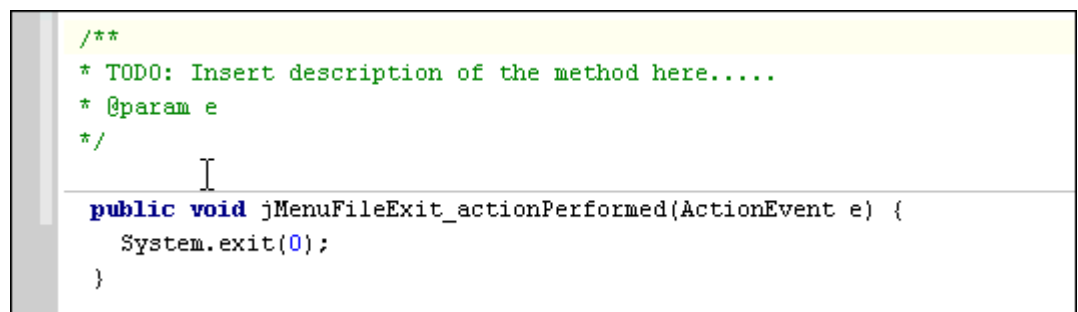


Figure 17 Code Generated by Easy.JavaDoc

You can select a member navigating through the list with the help of the usual keyboard. An Alternative way to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

On selecting the members press **Enter** to generate JavaDoc covering all of them.

### Options Dependency

---

You may adjust content of JavaDoc generated by [Easy.JavaDoc](#) using *Project Options / Productivity! / Easy.JavaDoc*. By default, [Easy.JavaDoc](#) generates `@return`, `@param` and `@throws` tags. You may also specify that it should generate `@author`, `@see` and `@since` tags. To enable or disable their generation, please open the *Project Options / Productivity! / Easy.JavaDoc* property page and select the appropriate check boxes. Please note that if you select generation of the `@author` tag, the [Easy.JavaDoc](#) inserts the tag's value as it is specified on the *Project Properties / General* property page.

In addition, on the same page you can specify the policy to be used by [Easy.JavaDoc](#) if JavaDoc comment already exists for a method or class. Based on your selection, [Easy.JavaDoc](#) can overwrite old comments, skip generation or ask your confirmation on comments rewriting. Note that these options may affect the members' list content in [Easy.JavaDoc.Insight](#) popup - if the option to skip members with existing JavaDoc is specified, all such members will be excluded from the members' list.

## Power Tools

---

Productivity! includes set of advanced tools those make Productivity! really unique. As all other Productivity! tools, these ones are intended to dramatically increase productivity of Java developer.

These tools include:

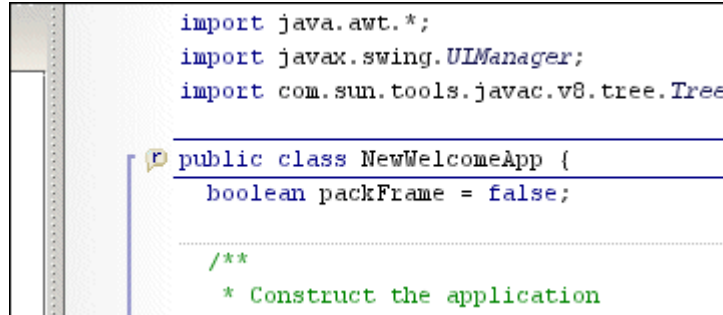
- [Rename Assistant](#) – unique tool that simplifies identifiers renaming by introducing "in-place" rename approach;
- [Assistants](#) – rich set of code and information assistants those provide quick information about code issues and, if possible, set of choices those allow problem resolving;
- [Task List](#) – the tool that allows viewing and managing list containing tasks, code issues and to-dos.
- [Smart.Templates](#) – the advanced code templates engine with ability to live update of related fields while typing.
- [Smart.JavaDoc](#) – the tool that allows JavaDoc authoring in mode is very close to WYSIWYG one. It is available as an additional viewer for Java file node.

### Rename Assistant - *Pro!*

---

Rename Assistant is a tool intended to simplify identifiers renaming. It introduces "in-place" editing approach and let the user to enter a new name and then choose whether hi/she likes to refactor, simply rename or skip changes of identifier name. To see Rename Assistant in action just place the caret to an identifier you wish to rename and start renaming it simply by typing new name. Rename Assistant pops up right in the place of your identifier; you can recognize it by the lines shown on the top and bottom of identifier being renamed and Rename Assistant icon placed on gutter. When new name is entered it's possible to:

- Press the Enter key to rename identifier using refactoring procedures.
- Press the Ctrl+Enter key to simply rename identifier.
- Press the Esc key or mouse click outside Rename Assistant to skip any changes.



```
import java.awt.*;
import javax.swing.UIManager;
import com.sun.tools.javac.v8.tree.Tree

public class NewWelcomeApp {
    boolean packFrame = false;

    /**
     * Construct the application
    */
}
```

Figure 18 Rename Assistant

It's possible to tune Rename Assistant behavior using the Editor Preferences | Productivity! | Assistant property page. There is ability to quickly turn assistant on/off using the Enable Rename Assistant button on the view toolbar.

Please note that Rename Assistant works only if there are no syntax errors in the source code.

#### Options Dependency

---

You can control Rename Assistant behavior using the *Editor Options | Productivity! | Assistant* property page.

### Assistant - *Pro!*

---

The [Assistant](#) tool provides visual feedback in the editor about any code issue by highlighting corresponding symbols using styled and colored line. By default, errors, warnings and To-Do's are highlighted using red, green and gray wavy line respectively. Each issue has description associated with it and description text for every issue can be shown in the hint window that appears if mouse cursor is placed under the code issue.

[Assistant](#) provides special Insight window, which can automatically pop-up in the location of code issue nearest to the caret position and shows information about the particular issue or list of possible actions to complete (resolve) it.

There are two kinds of [Assistants](#): [Info Assistant](#) and [Code Assistant](#).

The [Info Assistant](#) shows description for the issue if there are no actions that may be performed to resolve the issue. It is useful as it can provide more convenient feedback about the code issue than Structure View as it always visible and placed near caret and probably near the point of the user view.

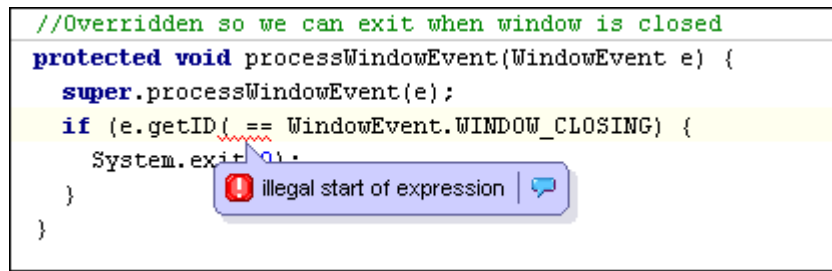


Figure 19 Info Assistant

The Code Assistant shows description for the issue along with the list of actions those can help to resolve it. Code Assistant provides most convenient way to resolve issues as it allows fixing of issue without having to leave current cursor position. It allows fixing of the issue manually or by invoking of appropriate tool. As soon as issue is resolved, it restores original caret position. The user is able to fix the issue quickly using prompted shortcut or just using mouse. There is also ability to popup and focus Code Assistant using *Alt-Enter* (CUA) keystroke and use usual keys to navigate through the fix actions and the *Enter* key to choose (invoke) required one.

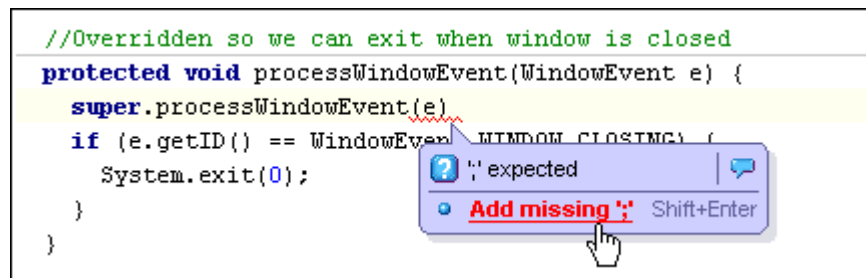


Figure 20 Code Assistant

Another way to fix issue is using context popup menu and this way is useful when Code Assistant is disabled.

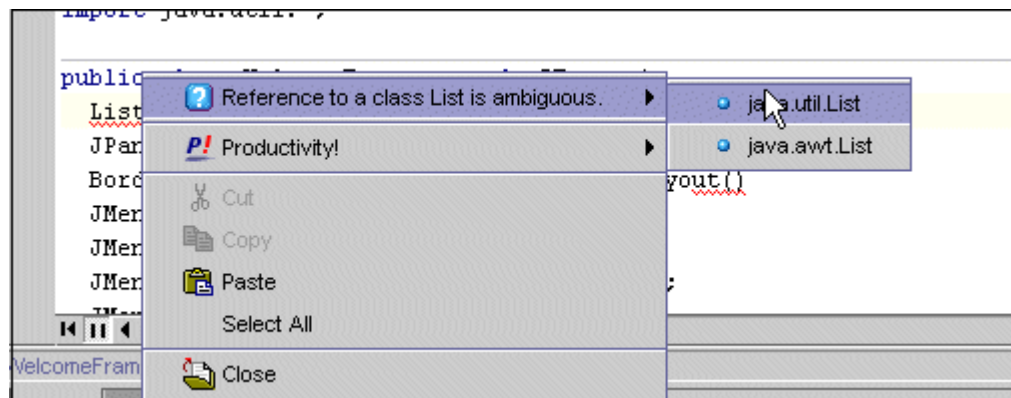


Figure 21 Code Assistant Integrated Into Context Menu

Assistant can be configured to automatically fix certain types of errors without prompting to the user. The following errors currently can be automatically fixed:

1. Type cast errors - by adding required type cast.

- Unknown class errors - by adding appropriate import statement. It's possible to specify policy for each short class name that exactly defines what action should be done to make this name known in the code.

It is usual thing when several classes with the same short names are be defined in different packages (java.util.List and java.awt.List is the good example). To simplify working with such classes it's possible to specify a list of most frequently used classes using the Project Properties | Productivity! | Assistant property page. Such list is used to determine particular class to import or to show in the top of the list in case of existence of several different candidates.

It's possible to disable assistants either globally (for all types of issues) using menu in the bottom right corner of [Thumbnail Gutter](#) or disable assistant for only specific group of issues.

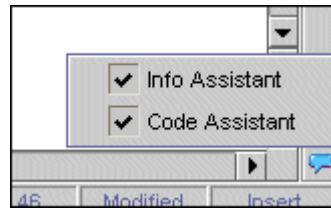


Figure 22 Disabling/Enabling Assistants' Menu

Below are several examples of [Code Assistant](#):

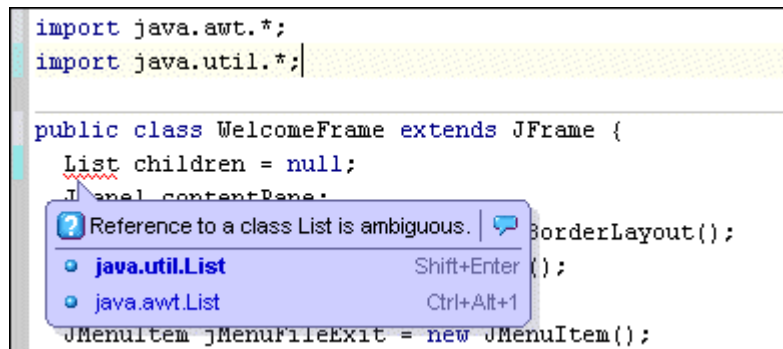


Figure 23 Ambiguous Names Issue Code Assistant

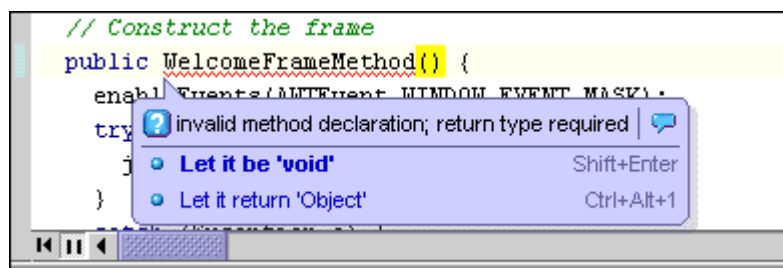




Figure 24 Invalid method declaration issue Code Assistant

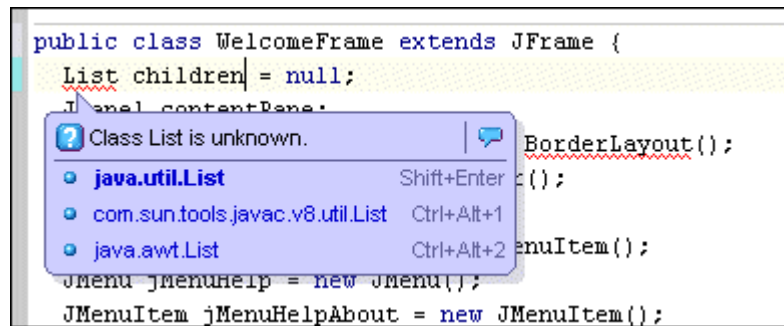


Figure 25 Unknown Class Issue Code Assistant

### Options Dependency

---

You can control Assistants behavior using the *Editor Options / Productivity! / Assistant* property page. The *Project Properties / Productivity / Assistant* property page can be used to specify project dependent Assistant options.

### Advanced To-Do's - *Pro!*

---

The *Advanced To-Do* tool expands the concept of To-Do comments by treating them as tasks rather than simple entities those can hold only textual information. The following attributes can be assigned to To-Do task:

- Priority;
- Completion Status;
- Owner;
- Start Date;
- Due Date;
- Reminder;
- URL.

All these attributes are encoded in the body of To-Do, so concept of *Advanced To Do's* doesn't require using of any additional storage. Tools those work with To-Do's e.g. *Advanced Java Structure View*) are aware of having such attributes and are able to correctly show only description of To-Do comment.

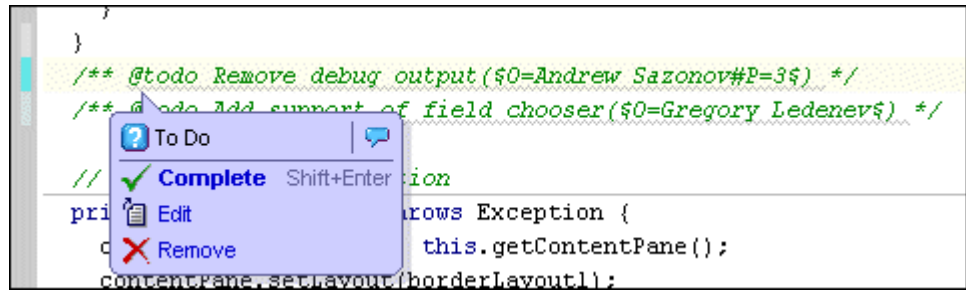


Figure 26 Advanced To-Do and Assistant

Any custom attributes can be directly added or changed manually in the source code according to the supported syntax.

The most convenient way of working with To-Do's is using **Task List**. The **Task List** allows navigation to a selected To-Do, changing of any To-Do's attributes (certainly excluding file URL) and removing the To-Do from the source file.

The **To-Do Assistant** allows execution of the most helpful actions for the To-Do at the caret position. Those actions include *Complete*, *Edit* and *Remove* the current To-Do task.

**Options Dependency**

You can control the behavior of **Assistants** using the *Editor Options | Productivity! | Assistant* property page.

**Task List - Pro!**

The **Task List** tool allows viewing and managing the list of tasks. **Task List** is embedded into JBuilder Message Pane. To show or hide **Task List** please use *View | Task List* menu item or appropriate menu item in editor context menu.

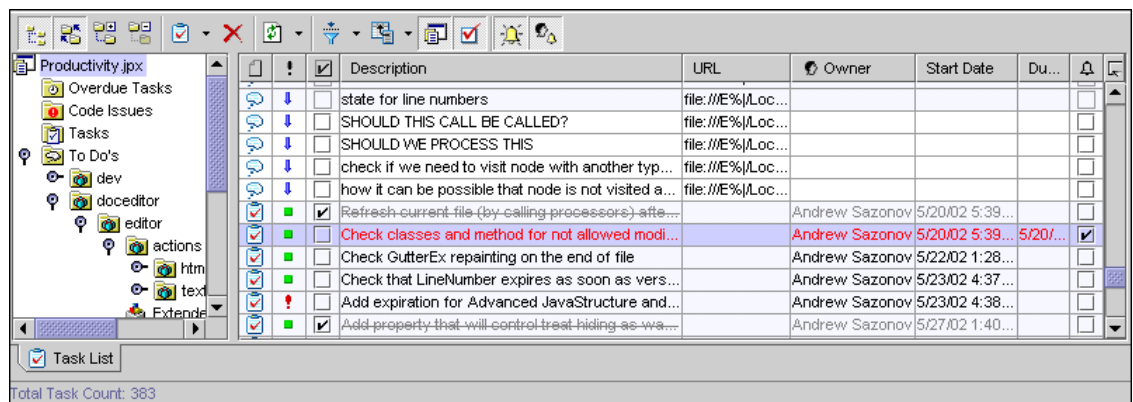


Figure 27 Task List Panel

## Introduction to Tasks' Concept

---

Each task is an entity that represents some work to be done.

The following attributes are belong to each task:

- Priority – can be one of the following: Low, Medium, High, Highest;
- Completion Status – specifies whether the task is completed or not;
- Description – specifies the description of the task;
- Owner – specifies the user name which owns the task;
- Start Date – specifies the date the task was/should be started;
- Due Date – specifies the date when the task should be done;
- Reminder Status – specifies whether reminder should be shown to the user about the task near to the due date.

The following task types are currently supported:

- Code Issues – errors and warnings collected from the file currently opened in the editor. They are interpreted as tasks because all errors and most of warnings should be fixed in order to successfully build source file;
- To-Do's – all @todo comments collected from the file currently opened in the editor and (optionally) collected for the whole project. All tasks collected for the whole project are cached in the file with ".todos" extension stored in the project folder. [Task List](#) tries to keep this cache up-to-date (as soon as file is opened) though there is ability to rebuild all of them manually;
- General Tasks – simple persistent tasks intended to general use. They are not related to any file but rather are related to the whole project. All those tasks are stored in the file with ".tasks" extension stored in the project folder.






## Task List User Interface

---

The [Task List](#) consists of toolbar which provides controls those allow maintaining and controlling [Task List](#) and Task List View which shows tasks. Task List View can be either Plain or Outlined one.

The Plain View shows the table in which each task occupies one row and each column represents appropriate single attribute of task. There is ability to resize, move or hide columns as well as to specify sorting order by clicking to desired column header.

Outlined View consists of Tasks Folder Tree View at the left and Plain View at the right of [Task List](#). Tasks Folder Tree View allows selection of tasks grouping and filtering criteria and it acts as "Master" while Plain View acts, as "Details" is their relationship. The following folders are available:













-  Project Folder – the root for all other folders;
-  Overdue Tasks –contains all overdue tasks;
-  Code Issues – contains code issues;
-  Tasks – contains General Tasks;
-  To Do's – contains all To-Do's outlined by packages of classes those To-Do's are belonging to.


It is possible to specify filters, sorting order, columns set, columns placement and sizes individually for each view and folder. All these settings are persistent between JBuilder sessions.

The [Task List](#) provides ability to synchronize selected task with file it is related to (if any). To open file associated with the task use double-click on the task in task list table.

The following table outlines actions available on the toolbar:

**Table 2 Task List Toobar Actions**

Icon	Description
	Toggles Plain/Outlined Views
	Specifies if tasks details view will include tasks from descendants of the currently selected package
	Expands all nodes in tree
	Collapses all nodes in tree
	Allows creation of new tasks
	Removes currently selected task/s
	Refreshes current view and optionally refreshes To-Do's for the whole project
	Allows applying filters for the current view
	Allows choosing visible columns for the current view.
	Specifies whether todo comments should be shown for the whole project or for active file only
	Specifies whether completed tasks should be visible
	Enables/Disables reminders tracking

	Specifies whether reminders should be shown only for tasks assigned to the current user

### Maintaining Tasks Using Task List

The **Task List** allows creation of new tasks, removal of existing ones and changing attributes of them. To change task attributes, **Task List** offers in-place editing capabilities and provides appropriate cell editor for the each attribute.

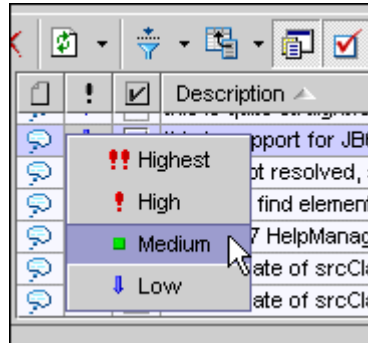


Figure 28 Changing Priority of the Task

The following table shows operations applicable to each task type.

Table 3 Task Operations

Task Type	Operation		
	New	Change Attributes	Remove
Code Issue	No	'Completed' attribute*	No
General Task	Yes	Yes	Yes
To-Do	Yes**	Yes**	Yes**

\* - This operation is applicable if there is at least one "Complete" action available for the particular Code Issue;

\*\* - These operations are applicable if Java file this To-Do is belonging to is being edited and this file is not read-only one;

Changing of 'Completed' attribute for To-Do's and General Tasks doesn't lead to any other actions rather than simple assigning of appropriate value to the task. This behavior is completely different for Code Issues as the nature of Code Issues is

different. To make Code Issue completed, the reason of it should be removed or the problem should be fixed somehow. Thus Code Issue can be completed if there are actions those can fix it and to complete it one of them should be executed as well.

For example, if code issue is “unknown class” or “ambiguous” one, it’s possible to select correct class from list of options, fix the issue and thus complete the task.

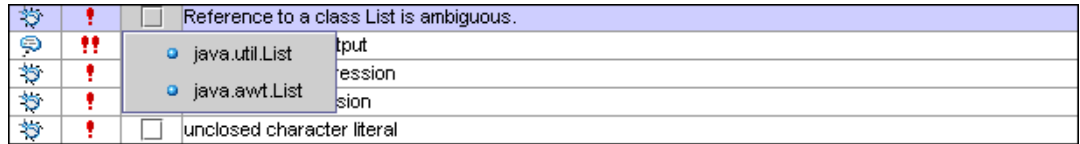


Figure 29 Fixing Code Issues Using Task List

### Reminders

There is ability to setup reminders for tasks which are not completed yet and have due date assigned. To setup reminder, the appropriate attribute of required task should be checked. The button should be pressed to enable reminders tracking. It is possible to instruct [Task List](#) to show reminders for tasks, which owned by current user only. To turn this mode on the button should be pressed.

The Reminder dialog is displayed when the time gap between current date and due date for particular task is less then specified value. In addition, there is an optional ability to play sound when reminder is occurring.

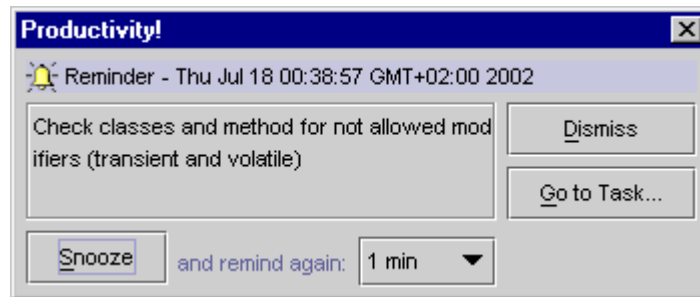


Figure 30 Reminder Dialog

The following actions are available from this dialog:

- Navigate to the particular task using the *Go to Task* button;
- Dismiss reminder using the *Dismiss* button;
- Snooze reminder for the certain period of time using the *Snooze* button and appropriate combobox.

The [Task List](#) hides all the shown Reminder dialogs for project dependent tasks when the user closes particular project or selects another one.

### Options Dependency

You can tune reminders behavior using the *IDE Options / Productivity* property page.

## Smart.Templates - *Pro!*

This tool introduces advanced templates concept and offers a lot of new possibilities and usability features. Smart.Templates may be invoked using usual **Ctrl+J** (CUA) shortcut or by typing template name and pressing **Tab** (by default) key. There is also possibility to expand last used template by pressing **Ctrl+Shift+J** (CUA) shortcut. This is useful if template is defined as one that may store and utilize user input collected during previous invocation.

The purpose of this tool is ability to maintain, easy find and paste frequently used code snippets (templates hereafter) to the currently editing document.

JBuilder built-in templates represents usual code fragment, which pastes “as is” and may be optionally formatted according to project code style (built-in templates only). Such a templates concept suits well for simple needs while leaving lots of manual modifications of pasted code for most of complicate cases.

**Smart.Templates** provides all functionality offered by build-in JBuilder templates and correctly indents the code fragment and optionally formats it according to current indent level and braces’ style. To minimize efforts needed to fit templates to the developers’ needs the concept of advanced templates is used. The main difference of this concept is ability to specify template fields inside the code snippet.

Each field is represented by its own name enclosed by **#** sign written right in the code snippet. In this case template acts as “live” running form rather than simple code fragment. On template expansion, all the code outside fields is pasted exactly as it is defined, while fields are represented by a set of editors those allows entering values using usual way. Each field “editor” has each own border and the focused one has the special border, which highlights it.

There is ability to navigate through all the fields using the **Tab** and **Shift+Tab** shortcuts. The **Enter** and **Shift+Enter** shortcuts allow navigating through the fields skipping ones with the same type.

```

public void processList(List aListToIterate)
{
    int size = aListT.size();
    for (int i = 0; i < size; i++) {
        Object object = (Object) aListT.get(i);
    }
}

```

Figure 31 Smart Template with Fields

It's possible to invoke another tools like Member Insight for selecting values of template fields.

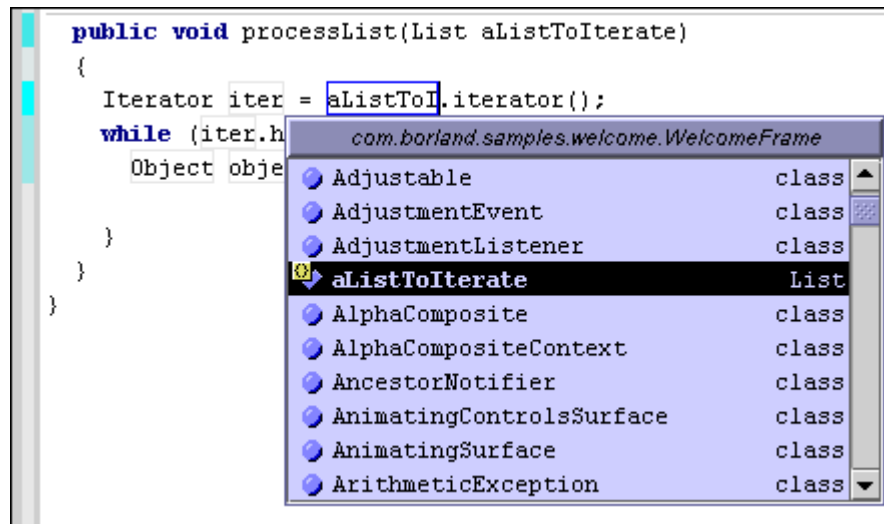


Figure 32 Entering Field Value Using JBuilder Member Insight

The following major features are applicable to template fields:

- All the template fields of same name are automatically synchronized that allow entering field value only once;
- Ability to specify default value or initialization expression for a template field. The default value or result of expression execution is automatically assigned to the field on template expansion;
- Ability to specify calculated expression for a template field. This allows dynamically calculate field value to reflect changes made by the user in this field or in the other ones;
- Ability to specify “change” expression for a template field. This allows making some actions after the field is changed and/or calculated.
- Ability to store values entered in the template fields and load previously stored values on the next template invocation. This allows to fill fields automatically during the consecutive template calls;
- Ability to utilize selected block of code as field value.

Expanded template enters the “running” state after invocation and user is able to fill template fields. A running template can be completed by pressing **Esc** key, changing code outside of any template fields (if appropriate option is turned on) or by filling all fields and using **Enter** key to navigate between them.

After the completion of running template, all the code written in the field “editor” is pasted directly to the document and the editor caret is optionally placed to the position specified by location of predefined field #|# (if one is included into template definition).

In addition, the [Smart.Templates](#) tool introduces the following features:



- Ability to shorten full-qualified names for classes stated in the template and adding appropriate import statements to the source file;
- Ability to use preprocessor instructions those allow dynamic building of template code using conditional and iteration statements;
- Ability to format template code according to the current project code style and current indent level;
- Ability to specify supported file types (Java, HML etc) for any template;
- Ability to specify supported context (e.g. "symbol", "comment" or "string") for any template;
- Ability to assign keyboard shortcut for any template. Shortcut assignments for different key maps are supported too.

### **Predefined Fields**

---

The following table outlines predefined template fields which always available to any template definition.

Table 4 Predefined Template Fields

Field Name	Description
	Specifies the location should be used for caret after template completion
selBlock	Inserts content of the code block has been selected before the template expansion.
selBlock.asComment	Inserts a code block has been selected before the template expansion and transforms it into a Java block comment.
selBlock.asString	Inserts a code block has been selected before the template expansion and correctly transforms it to a Java string.
currentDate	Inserts current date string.
currentDateTime	Insert current date and time string.

### Expressions

---

The most powerful feature of [Smart.Templates](#) is ability to specify expressions needed to be evaluated during template execution. This allows developing truly “smart” and dynamic templates those can be easy adapted to developers’ needs. In general, any expression should conform to expressions’ definition rules in Java language and can contains only the following lexemes:

- String, character and numeric literals;
- + - \* / operators;
- Variables;
- Functions calls;
- Open and close parenthesis;

The [Smart.Templates](#) provides automatic types conversion during evaluation of expressions so expression can contains literals, variables and functions with different types and/or return types. If possible, all these different types will be correctly cased.

The following functions are available:

#### Utility Functions

---

Functions are included into this group are intended to suit miscellaneous utility purposes.

String <a href="#">getLocalVariableName</a> (String aType)
--

Composes variable name based on passed variable type.

**Parameters:** `aType` – string that represents variable type

**Return:** name for variable based on variable type

String `getVariableInitValue`(String aType)

Composes variable initialization value based on passed variable type.

**Parameters:** `aType` – variable type

**Return:** variable initialization value. It will be `-1` for numeric types, `false` for boolean and `null` for any other Java type.

#### Template Functions

Functions are included into this group allow gaining access to any field within running template.

String `getFieldValue`(String aName)

Returns current value of the template field

**Parameters:** `aName` – name of the field

**Return:** value of the field; null if field with passed name is not found

void `saveFieldValue`(String aName)

Saves field value as string to allow it future use

**Parameters:** `aName` – name of the field

void `saveFieldValueAsInteger`(String aName)

Saves field value as integer to allow it future use

**Parameters:** `aName` – name of the field

Object `loadFieldValue`(String aName, Object aDefaultValue)

Obtains previously saved field value

**Parameters:**

`aName` – name of the field  
`aDefaultValue` – optional default value

**Return:** previously saved field value; if field value was not saved before it returns `aDefaultValue` or null if default value is not specified.

### System Functions

Functions from this group provide access to some of system properties and resources.

String `getSystemProperty`(String `aName`)

Obtains the value of system property

**Parameters:** `aName` – name of system property to get

**Return:** value of system property; null if property with such name is not found

String `getDate`(String `aFormat`)

Obtains string representation of current date according to optional format

**Parameters:** `aFormat` – optional parameter that defines the form of date output. The valid values are "SHORT", "MEDIUM", "LONG", "FULL".

**Return:** string representation of current date

String `getDateTime`(String `aDateFormat`, String `aTimeFormat`)

Obtains string representation of current date and time according to optional format

**Parameters:**

`aDateFormat` – optional parameter that defines the form of date output. The valid values are "SHORT", "MEDIUM", "LONG", "FULL".

`aTimeFormat` – optional parameter that defines the form of time output. The valid values are "SHORT", "MEDIUM", "LONG", "FULL".

**Return:** string representation of current date

### String Functions

Functions from this group provide different string manipulation routines.

String `toUpper`(String `aString`)

Converts passed string to upper case.

**Parameters:** `aString` – string need to be converted

**Return:** string converted to upper case

String `toLowerCase`(String `aString`)

Converts passed string to lower case.

**Parameters:** `aString` – string need to be converted

**Return:** string converted to lower case

String `toggleCase`(String `aString`)

Converts passed string so each character of it became toggled from lower case to upper one and vice versa.

**Parameters:** `aString` – string need to be converted

**Return:** converted string

String `toConst`(String `aLiteralValue`)

Composes name for string literal based on passed literal value. It is composed as "C" style constant where locations of underscores are match to locations of characters in upper case in the literal value.

**Parameters:** `aLiteralValue` – string literal value

**Return:** name for string literal based on literal value

String `fromConst`(String `aLiteralName`)

Composes value for string literal based on passed literal name using the same rules as for composing name for literal based on its value.

**Parameters:** `aLiteralName` – string literal name

**Return:** value for string literal based on literal name

String `capitalize`(String `aString`)

Capitalizes the first letter of passed string.

**Parameters:** `aString` – string to capitalize

**Return:** capitalized string value

String `deCapitalize`(String `aString`)

Decapitalizes the first letter of passed string.

**Parameters:** `aString` – string to capitalize

**Return:** decapitalized string value

String `subString`(String `aString`, String `aStringToFind`)

Obtains a new string by trimming the right part of passed one starting from occurrence of another string.

**Parameters:**

`aString` – string to be trimmed

`aStringToFind` – string to find

**Return:** trimmed string value

String `getConstPrefix`(String `aName`)

Obtains the prefix from the passed literal name.

**Parameters:** `aName` – name of literal

**Return:** obtained prefix or passed literal name if prefix can't be obtained

**Project and  
Editor  
Dependant  
Functions**

Functions included into this group provide access to some properties of the active project and to information about context available for the caret position of the editor.

String `getProjectProperty`(String `aName`)

Obtains the value of property of active project

**Parameters:** `aName` – name of project property to get

**Return:** value of project property; null if property with such name is not found

String [getFullName\(\)](#)

Obtains the name of Java class at caret position

**Return:** full-qualified name of class

String [getClassName\(\)](#)

Obtains the short name of Java class at caret position

**Return:** short name of class

String [getPackageName\(\)](#)

Obtains the package name of Java class at caret position

**Return:** package name

---

### **Smart.Templates.Insight**

This tool allows choosing a template to expand. When editing a file, press **Ctrl+J** (CUA) to invoke [Smart.Templates.Insight](#). The [Smart.Templates.Insight](#) popup will be shown with the list of template matching a word at the cursor position. The list may be empty if there are no matching templates though. To find matches, type a word in the Template edit box and [Smart.Templates.Insight](#) will dynamically rearrange the list of templates to show the matching ones. You can also leave the Template edit box blank to view all templates.

You can select a template navigating through the list with the help of the usual keyboard. An alternative way to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

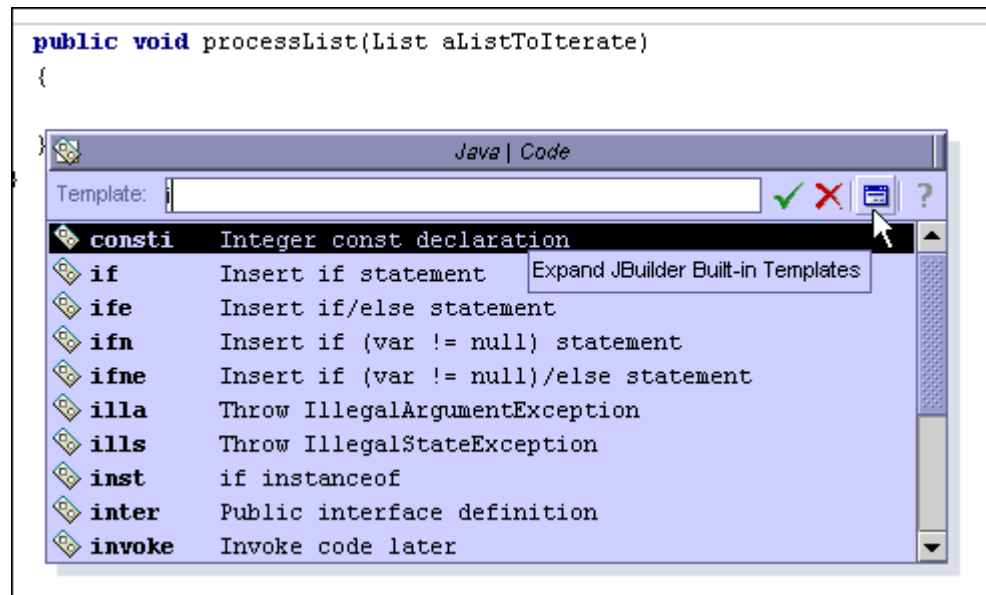


Figure 33 Smart.Templates.Insight Popup Window

Press the **Enter** key when you select the template and **Smart.Templates.Insight** will expand selected one.

#### “On the Fly” Smart Templates

The main idea of such templates is ability to quickly generate and use template definition based on the block of text. All the repetitive tokens from the block those are not predefined ones will be replaced by the template fields definitions.

This approach allows using templates those need one or several times only without having to manually manage template definitions.

```

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = frame.getSize();
if (frameSize.height > screenSize.height)
    frameSize.height = screenSize.height;
if (frameSize.width > screenSize.width)
    frameSize.width = screenSize.width;
frame.setLocation((screenSize.width - frameSize.width) / 2, (screen

```

Figure 34 Selected Code Fragment will be Copied



```
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = frame.getSize();
if (frameSize.height > screenSize.height)
    frameSize.height = screenSize.height;
if (frameSize.width > screenSize.width)
    frameSize.width = screenSize.width;
frame.setLocation((screenSize.width - frameSize.width) / 2, (scre
```

Figure 35 The Same Code Fragment Inserted Using "On The Fly" Templates

To invoke "On The Fly" template based on the clipboard content please use **Alt+Shift+J** (CUA) shortcut. Another option to invoke "On The Fly" template is using [Clipboard.Insight](#). You just need to select required item from the local clipboard list and press **Ctrl+Enter** shortcut to invoke template based on it.

### Options Dependency

---

You can control [Smart.Templates](#) behavior and manage the template list using the *Editor Options | Productivity | Smart.Templates* property page.

**Smart.JavaDoc - Pro!**

Creation and editing of source code documentation is an important part of the development process. Unfortunately, JBuilder provides very basic support for this so developer is forced to create the entire JavaDoc comments markup manually.

To significantly increase productivity of source code documentation creation, to reduce amount of possible JavaDoc errors those can occur during this process and to insure that documentation is still valid after performed refactoring, Productivity! offers very powerful tool intended to provide visual JavaDoc editing.

The [Smart.JavaDoc](#) tool offers rich JavaDoc creation, navigation and editing functionality. It represents additional viewer for Java files and is accessible via appropriate tab at the bottom of the editor.

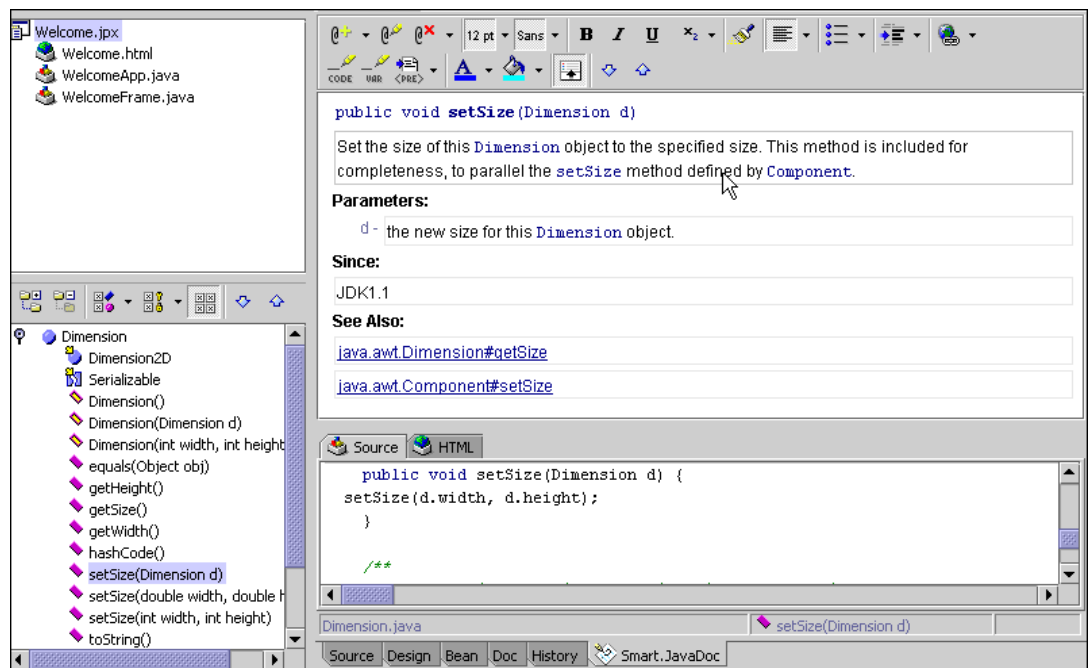


Figure 36 Smart.JavaDoc Overview

The major features of [Smart.JavaDoc](#) are:

- Close to WYSIWYG mode of JavaDoc comment editor (according to HTML output provided by standard JavaDoc doclet);
- Rich HTML editing capabilities ([Smart.JavaDoc](#) does not support whole set of HTML tags. However, the supported set is quite enough for creation of professional quality documentation);
- Ability to discover and visually highlight JavaDoc conflicts and errors (such as unsupported tags, parameters and throws conflicts, missing tags etc);
- Ability to easily correct found JavaDoc conflicts and errors;

- Rich functionality of JavaDoc comment structure management (adding specific tags, adding all required tags, removing all invalid and unused tags, tags renaming may be performed in couple of clicks);
- Convenient functionality for navigation and instant accessing members for which JavaDoc comments should be created;
- Ability to filter members used for documentation creation via advanced Java Structure Component;
- Ability to perform two-way editing of JavaDoc comment – using both Smart.JavaDoc and usual Java source editor;
- Ability to browse source code while editing JavaDoc;
- Ability to instantly get preview of JavaDoc comment will be generated;
- Sophisticated multilevel undo/redo support;

### Smart.JavaDoc User Interface

---

The [Smart.JavaDoc](#) tool is an additional viewer available for Java files.

It consists of two major parts:

- Java Structure View, which allows you easily navigate between members for which JavaDoc comment should be generated;
- JavaDoc View, intended to provide JavaDoc comment editing and previewing;

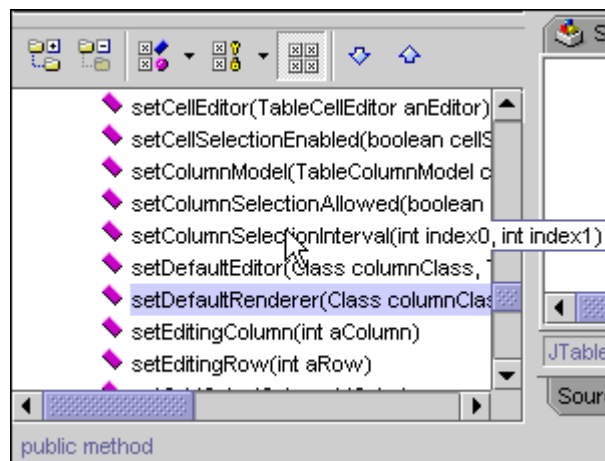


Figure 37 Smart.JavaDoc Java Structure View

The [Java Structure View](#) offered by [Smart.JavaDoc](#) is similar to one provided for Java files. The main difference is that all unimportant and not suitable members are filtered out from its view. There are two additional actions, those allows to navigate to the next/previous member of [Java Structure View](#). These actions may be invoked using **Ctrl+Page Down** and **Ctrl+Page Up** shortcuts, respectively. Please note that these shortcuts are operational even if [Java Structure View](#) is not visible.

The JavaDoc View contains toolbar with set of actions used for JavaDoc editing, the JavaDoc editor and, optionally, the preview panel.

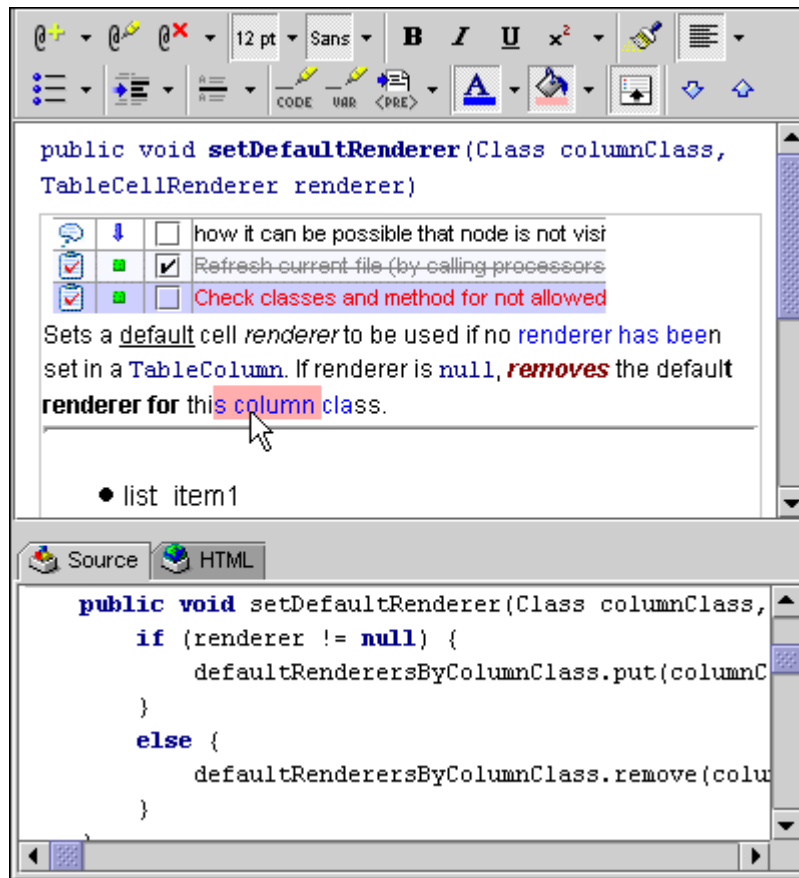



Figure 38 Smart.JavaDoc View with Source Code View

There is ability to showing/hiding of the preview panel using the  button on the Smart.JavaDoc toolbar.

The preview panel consists of two tabs. The *Source* one allows you quickly view source code for the currently edited member as well as for the whole file.

The *HTML* tab contains live preview of the JavaDoc comment that will be generated for current member.

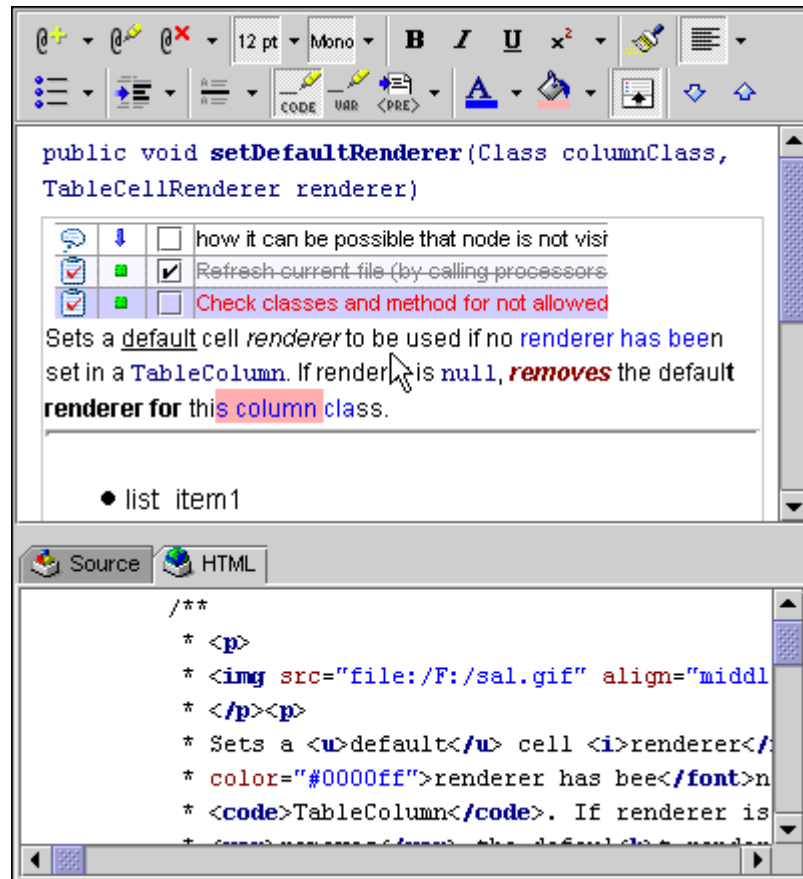


Figure 39 Smart.JavaDoc View with JavaDoc Comment Preview

The JavaDoc editor panel includes set of editors (one editor per one JavaDoc tag). Each editor or editors' group is labeled with the name of the tag it belongs to. To navigate to the next/previous tag editor, the **Tab** (or **Ctrl+Down**)/**Shift+Tab** (or **Ctrl+Up**) shortcuts are provided respectively.

### JavaDoc Editing Using Smart.JavaDoc

The **Smart.JavaDoc** tool offers rich functionality for WYSIWYG editing of the JavaDoc comment in form close to one will be generated by standard JavaDoc doclet and, in turn, rendered by Help Viewer.

Editing of JavaDoc is started by activation of **Smart.JavaDoc** using appropriate tab at the bottom of the editor. **Smart.JavaDoc** first tries to discover member in the current caret position and offers editing of JavaDoc comment for it. If there are no JavaDoc comment exists for the current source code member, the default JavaDoc template for all suitable tags is offered. If JavaDoc comment exists, **Smart.JavaDoc** will show editors only for tags defined in it.

Changes made by user while editing the JavaDoc comment are written to the source file on closing the **Smart.JavaDoc** tab, on selection of another member (class, field or method) or by saving the document.

Please note that `Smart.JavaDoc` may perform conversion of some unsupported tags found in JavaDoc comment.

The following are limitations of `Smart.JavaDoc` regarding support of HTML tags:

1. `Smart.JavaDoc` doesn't support the following HTML tags so it skips and ignores them

`<base>`, `<basefont>`, `<body>`, `<html>`, `<title>`, `<meta>`, `<script>`, `<style>`, `<head>`, `<applet>`, `<object>`, `<frame>`, `<noframes>`

2. The following tags are partially supported by `Smart.JavaDoc` – they will be displayed but in some cases they may be edited incorrectly:

`<form>`, `<input>`, `<option>`, `<textarea>`, `<table>`, `<td>`, `<tr>`, `<th>`

3. There is set of tags those will be replaced by equivalent ones:

Original tag	Replacing Tag
<code>&lt;em&gt;</code>	<code>&lt;i&gt;</code>
<code>&lt;strong&gt;</code>	<code>&lt;b&gt;</code>
<code>&lt;address&gt;</code>	<code>&lt;i&gt;</code>
<code>&lt;cite&gt;</code>	<code>&lt;i&gt;</code>
<code>&lt;small&gt;</code>	<code>&lt;font size=10&gt;</code>
<code>&lt;big&gt;</code>	<code>&lt;font size=18&gt;</code>
<code>&lt;div&gt;</code>	<code>&lt;p&gt;</code>
<code>&lt;dt&gt;</code>	<code>&lt;p&gt;</code>
<code>&lt;blockquote&gt;</code>	<code>&lt;p&gt;</code>
<code>&lt;tt&gt;</code>	<code>&lt;code&gt;</code>

4. `Smart.JavaDoc` does not support nesting of container tags like `<p>`, `<pre>`, `<ul>`, `<ol>`, `<div>`, `<dt>`, `<table>`. If JavaDoc comment to be edited contains such nested structures, they will be translated to linear ones, if possible;
5. If the color `attributes` are not set for the following tags, `Smart.JavaDoc` artificially adds them to allow visual recognition of the tags content. These artificial colors are temporary ones and will not appear in the resulting JavaDoc.

Tag	Color Assigned
<pre>	Green
<var>	Maroon
<anchor>	Blue
<code>	Navy

6. Smart.JavaDoc doesn't support paragraph align attributes for the following tags:

<pre>, <ul>, <ol>, <table>

Also, please note that Smart.JavaDoc considers all words starting from @ as JavaDoc tags.

### Smart.JavaDoc Toolbar

Most of the actions accessible for JavaDoc editing are provided by Smart.JavaDoc toolbar.

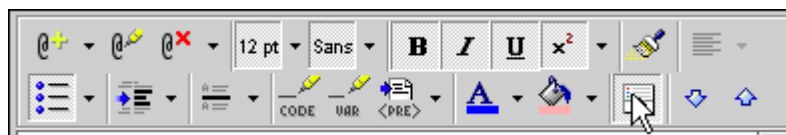


Figure 40 Smart.JavaDoc Toolbar

The following groups of actions can be found in the Smart.JavaDoc toolbar:

- *Tags* –actions group provides ability to manipulate the JavaDoc structure by adding, editing or removing particular JavaDoc tags.

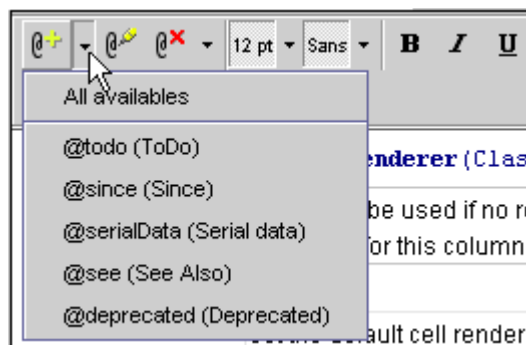


Figure 41 Adding Required Tags

- *Font* - actions group provides ability to apply appropriate font size and family to the currently selected block of text;

- *Font Style* –actions group provides ability to apply appropriate font style (bold, italic, underline and subscript/superscript ones) to the currently selected block of text;



Figure 42 Applying Subscript Style

- *Format Painter* – a tool is similar to one can be found in Microsoft Word. Allows performing of quick copying of formatting style from one text fragment to another one;
- *Paragraph Alignment* –actions group provides ability to set alignment of paragraph;
- *Bullets and Numbering* –actions group provides ability to convert currently selected code fragment to a list (either numbered or unnumbered);



Figure 43 List Style Selection

- *Indents* –actions group that provides ability to increase or decrease the indent level of the current paragraph;

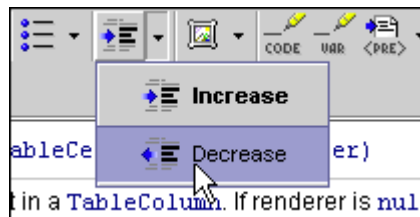


Figure 44 Controlling Indent Level

- *Insert Objects* –actions group that provides ability to insert various HTML objects.



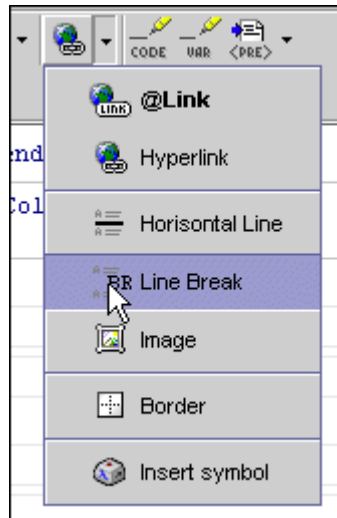


Figure 45 Inserting Special Objects

The following objects are supported:

- a. *Link* - corresponds to the JavaDoc @link tag. The user is able to specify properties of a new link or change them for existing one by using *Link Properties Dialog*

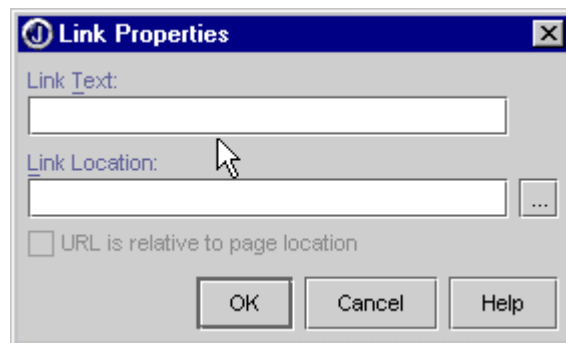


Figure 46 Link Properties Dialog

- b. *Hyperlink* – corresponds to HTML `<a href=...>` tag;
- c. *Horizontal line* – corresponds to HTML `<hr>` tag;
- d. *Line Break* - corresponds to HTML `<br>` tag;
- e. *Border* – applies appropriate HTML style to the current paragraph;
- f. *Image* – corresponds to HTML `<img>` tag. The user is able to specify properties of a newly created image or change them for existing one using the *Image Properties Dialog*

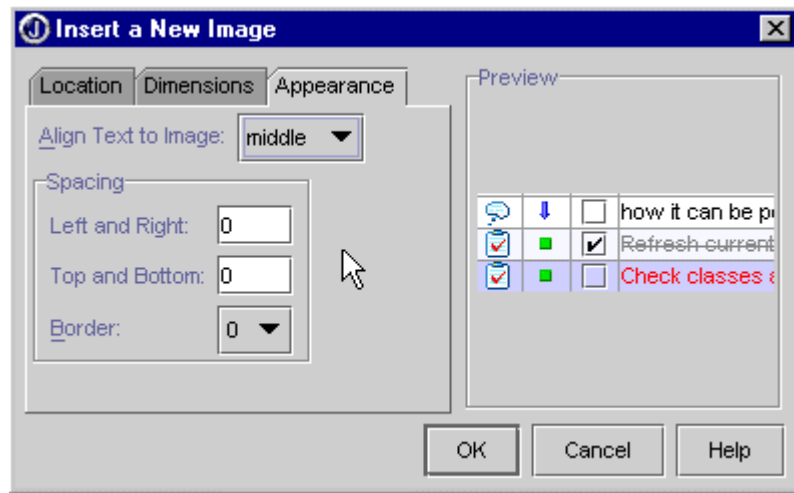


Figure 47 Image Properties Dialog

- g. *Insert Symbol* – allows to insert special symbol to JavaDoc comment using the *Insert Special Symbol* dialog

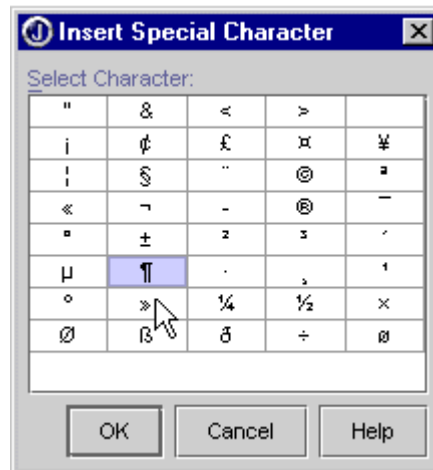


Figure 48 Insert Special Character Dialog

- *Colors* –actions group which provides ability to specify foreground and background color for the selected fragment of code;
- *Options* – provides ability to enable/disable displaying of the preview panel;
- *Navigation* - includes two actions those allow navigation to the next/previous member of Java file. These actions may be invoked using **Ctrl+Page Down** and **Ctrl+Page Up** shortcuts, respectively.
- *Block Style* –actions group which provides ability to specify style of the text block – paragraph (**<p>**), preformatted block (**<pre>**), code (**<code>**) or variable width (**<var>**);

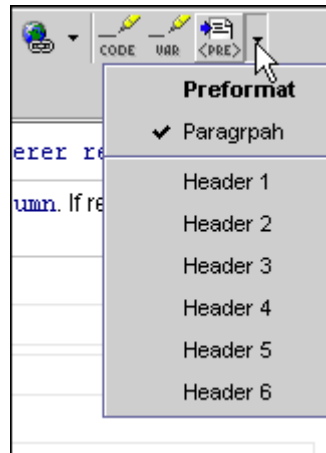


Figure 49 Text Block Style Selection

There is ability to specify which actions group should be visible on the [Smart.JavaDoc](#) toolbar using the toolbar context menu.

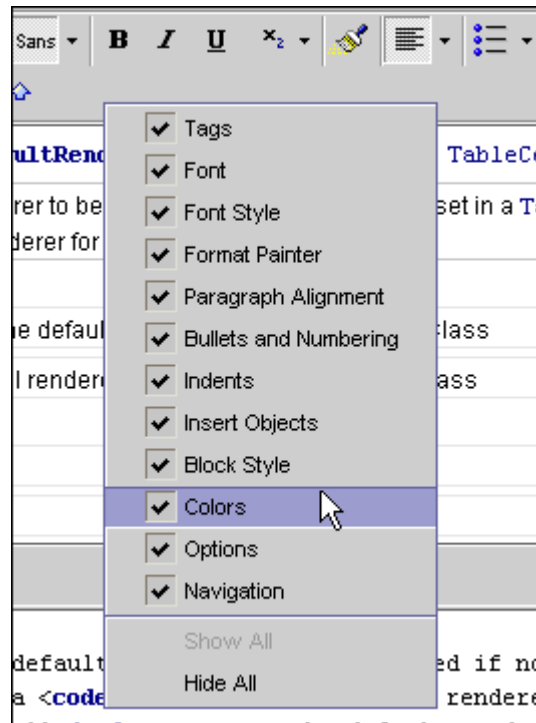


Figure 50 Controlling Smart.JavaDoc Toolbar

### JavaDoc Errors Highlighting

---

The [Smart.JavaDoc](#) tool provides rich functionality that allows detecting whether errors or conflicts between Java code definition and corresponding JavaDoc exist.

If some error or conflict is detected, [Smart.JavaDoc](#) highlights appropriate element using the underline with style and color corresponding to the issue priority. The reason

of the highlighted issue can be found in the hint that appears when mouse cursor is placed over the highlighted element.

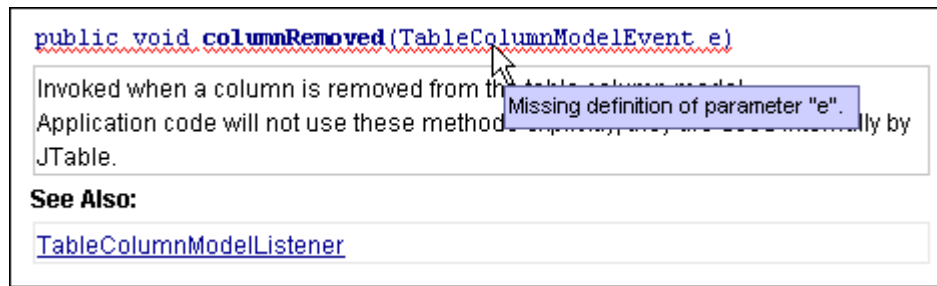


Figure 51 JavaDoc Comments Errors Detection

The style of the issue highlight is defined by settings can be found on the *Editor Options / Productivity! / Smart.JavaDoc* property page.

You can easily fix found errors utilizing appropriate commands available from the popup menu or from toolbar. Using them you can add, remove or edit required JavaDoc tags.

### Smart.JavaDoc Shortcuts

There are a lot of actions provided by [Smart.JavaDoc](#) those are accessible using keyboard shortcuts. The following table summarizes these actions along with corresponding shortcuts.

Table 5 Smart.JavaDoc Key Bindings

| Action  | Shortcut         |
|---|------------------|
| Copy to clipboard                                       | Ctrl+C           |
| Cut to clipboard  | Ctrl+X           |
| Paste from clipboard                                    | Ctrl+V           |
| Select all  | Ctrl+A           |
| Insert new paragraph / Insert new line in the PRE block | Enter            |
| Word left   | Ctrl+Left        |
| Word right  | Ctrl+Right       |
| Select word left  | Ctrl+Shift+left  |
| Select word right                                       | Ctrl+Shift+Right |
| Go to start of line                                     | Home             |
| Go to start of document                                 | Ctrl+Home        |

|   |  |
|---|--|
| Go to end of line   | <b>End</b>   |
| Go to end of document   | <b>Ctrl+End</b>  |
| Select to line start  | <b>Shift+Home</b>  |
| Select to document start  | <b>Shift+Ctrl+Home</b>   |
| Select to line end  | <b>Shift+End</b>   |
| Select to document end  | <b>Shift+Ctrl+End</b>  |
| Go to line above  | <b>Up</b>  |
| Go to line below  | <b>Down</b>  |
| Select to line above  | <b>Shift+Up</b>  |
| Select to line below  | <b>Shift+Down</b>  |
| Increase paragraph indent   | <b>Ctrl+Tab</b>  |
| Decrease paragraph indent   | <b>Ctrl+Shift+Tab</b>  |
| Go to the next tag  | <b>Tab or Ctrl+Down</b>  |
| Go to the previous tag  | <b>Shift+Tab or Ctrl+Up</b>  |
| Go to the next member   | <b>Ctrl+Page Up</b>  |
| Go to the next member   | <b>Ctrl+Page Down</b>  |
| On link or image – activate property action<br>Otherwise – select paragraph | Mouse-double-click   |
| Set selected text BOLD/DEFAULT  | <b>Ctrl+B</b>  |
| Set selected text ITALIC/DEFAULT  | <b>Ctrl+I</b>  |
| Set selected text UNDERLINE/DEFAULT   | <b>Ctrl+U</b>  |
| Insert Horizontal ruler ( <b>&lt;hr&gt;</b> )                               | <b>Ctrl+L</b>  |
| Insert line break ( <b>&lt;BR&gt;</b> )                                     | <b>Ctrl+Enter</b>  |
| Format painter  | Mouse click – to copy format,<br>mouse click to paste format, Esc or<br>click on button to clear |

## Editor Enhancements

---

The overall productivity of developer greatly depends on source code editor. That's why Productivity! includes several tools those enhances built-in JBuilder editor and allows to gain Productivity! users even higher level of productivity.

These tools may be separated on several categories:

- Code editing improvements - [Smart.Clipboard](#), [Auto.Indent](#), [Smart.Selection](#), [Smart.Braces](#) tools;
- Matching elements highlights - [Smart.Braces.Highlight](#), [Matching.Code.Highlight](#) tools;
- Usability improvements - [Thumbnail Gutter](#), [Smart.Gutter](#), [Advanced Text View Status Bar](#), [Line Numbers](#) tools;
- Editor look and feel improvements - [Methods and Classes Separator](#), [Changes Highlight](#), [Classes Highlight](#) tools;

### Smart.Clipboard - *Pro!*

---

This tool introduces the replacement for standard clipboard actions and offers a number of improvements and new features.

#### Paste Action

---

In general, this action acts as usual Paste action but, in addition, also allows making the following actions:

- Auto indentation of Java code during paste operation according to the indent level at the point of pasting;
- Automatically insertion of appropriate import statements on paste for classes used in the copied fragment (if clipboard content was copied from JBuilder editor with Java code).

#### Copy/Cut Actions

---

In general, those actions acts as usual Copy/Cut actions but they allows making the following additional actions:

- Collecting information about classes used in the copied/cut code fragment.
- Storing the copied/cut code fragment in the local clipboard queue that allows future use of several code fragments.

#### Swap Action

---

This action allows swapping the content of the clipboard with currently selected block of code. It can be invoked using the **Ctrl+Shift+Insert** (CUA) shortcut.

## Pop Paste Action

---

This action allows consecutive popping and pasting of code fragments from the local clipboard history in the LIFO order. It can be invoked using the **Ctrl+Alt+Insert** (CUA) shortcut.

## Clipboard.Insight

---

This tool allows viewing of local clipboard queue and pasting one or several selected fragments in the editor. When editing a file, press **Alt+Shift+V** (CUA) to invoke **Clipboard.Insight**. The **Clipboard.Insight** popup will be shown with the list of code fragments copied/cut to clipboard during the JBuilder session.

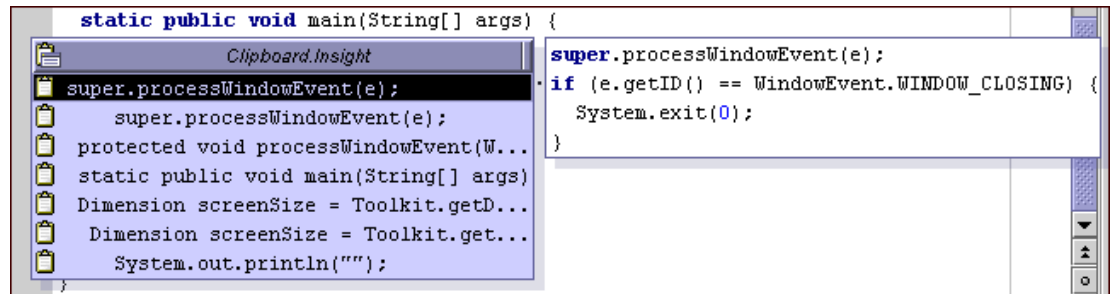


Figure 52 Smart.Clipboard.Insight Window Along with Clipboard Content Popup

You can select a code fragment navigating through the list with the help of the usual keyboard.

**Clipboard.Insight** list supports multiselection feature, thus it's possible to paste several items at once. To do this, select the code fragments should be pasted using the **Ctrl** key and mouse and then press the **Enter** key - **Clipboard.Insight** will insert all selected items.

It's possible to control the order of inserted items. Holding **Chift** while pressing **Enter** will lead to inserting selected items in reverse order.

## Options Dependency

---

You can control the behavior of **Smart.Clipboard** using the *Editor Options / Productivity! / Tools* property page.

## Smart.Selection - *Pro!*

---

The **Smart.Selection** tool represents set of several selection enhancement actions those allow to simplify selection operations while editing Java files. The following actions are provided:

- Selection of the whole code block, statement, method or class. These actions are available in the editor context popup menu as well as in the JBuilder menu bar. There are no default key bindings for these actions but you are able to easy assign them using the Key Map editor.

- Expanding Selection **Ctrl+W** (CUA) and Narrowing Selection **Ctrl+Shift+W** (CUA). These actions are also available in editor context popup menu and in the JBuilder menu bar as well. They allow to expand/narrow current selection incrementally to outer/inner source element respectively. The approximate order of selections is as follows (the exact one depends on current structure of the Java code and current caret position):
  - Word under cursor;
  - Expression;
  - Statement;
  - Code block
  - Enclosing statements and code blocks;
  - Method;
  - Class;
  - Whole file.

### Smart.Gutter - *Pro!*

The **Smart.Gutter** is a gutter placed at the left side of the editor (and right from JBuilder editor gutter) and it is intended to show miscellaneous hints concerning corresponding Java code by arranging appropriate gutter marks. The **Smart.Gutter** allows viewing tool tips those show description of a **Smart.Gutter** mark when mouse cursor is moving over it. Optionally, any **Smart.Gutter** mark can provide operations applicable to the corresponding code. If there is at least one operation available, the ↑ sign is shown at the left of a gutter mark. In this case, mouse cursor is changed to the hand one and the operations can be executed by mouse button click. The most typical operation is navigation to some piece of code related somehow to the code corresponding to the mark.

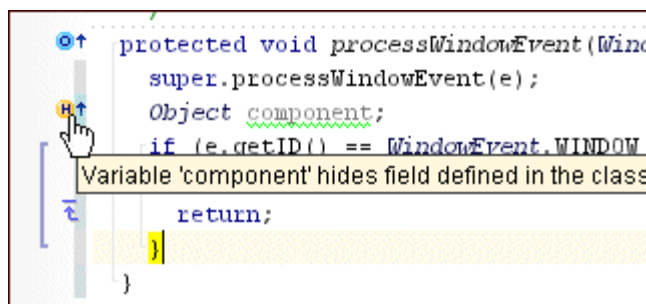



Figure 53 Smart.Gutter

The **Smart.Gutter** marks are currently supported for the following cases:

-  - There is method overriding;








-  - There is declaration of constructor with the same signature as the one defined in the super class.
-  - There is a break statement.
-  - There is a continue statement.
-  - There is a return statement.
-  - There is a throw statement.



Figure 54 Smart.Gutter Marks Show/Hide Button

There is ability to turn **Smart.Gutter** marks on/off using the  button in the View Toolbar.

### Thumbnail Gutter - *Pro!*

---

The **Thumbnail Gutter** is an additional gutter at the right side of the view. It's intended to quickly provide information about state of the currently edited Java source file. Unlike to the built-in JBuilder gutter, **Thumbnail Gutter** always shows gutter marks for the whole source file. Supported gutter marks are errors, warnings and To Dos.



Figure 55 Thumbnail Gutter

In addition, the gutter allows quick navigation to a corresponding code issue by single mouse button click.

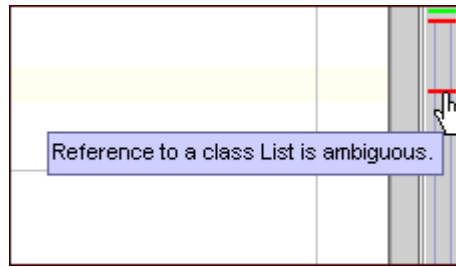


Figure 56 Thumbnail Gutter with Hint for Error

This gutter shows distribution of code issues along the whole file. Also, it allows quickly get information about particular mark using hint that appears if mouse cursor is placed over appropriate gutter mark.

There are several additional features provided by [Thumbnail Gutter](#):

- Ability to navigate to any position within file using mouse button double-click in the desired location on the gutter;
- Status icon at the top of the gutter, which shows the status of the file. There are following statuses:
  - OK - no errors or warnings found;
  - Errors – any errors found;
  - Warnings - any warnings found;
  - Running – Code is being analyzed.
- Assistant icon at the bottom of the gutter, which provides access to Assistants menu that allows controlling of [Code Assistant](#) and [Info Assistant](#) status;
- Ability to display bounds of the class which caret position belongs to. This feature allows instantly understand whether errors and warnings exist those are related to that class.

### Classes Highlight - *Pro!*

---

Classes Highlight is a tool intended to highlight classes used in the code. It uses "Extra keyword" style so to see it in action please customize this style (to make it different from "Identifier" style) using the Editor Preferences | Editor | Color | Java property page.

```

public class WelcomeApp {
    boolean packFrame = false;

    /**
     * Construct the application
     */
    public WelcomeApp() {
        WelcomeFrame frame = new WelcomeFrame();
        //Pack frames that have useful preferred si
    }
}
    
```

Figure 57 Classes Highlight

By default all (\*) classes are highlighted but it's possible to specify which classes should be highlighted (e.g. java.\*;javax.\*;) for each project using the *Project Properties / Productivity / Tools* property page.

**Advanced Text View Status - *Pro!***

Productivity! includes enhanced editor status bar intended to provide more information about the current status of file in the editor as well as to simplify navigation operations.

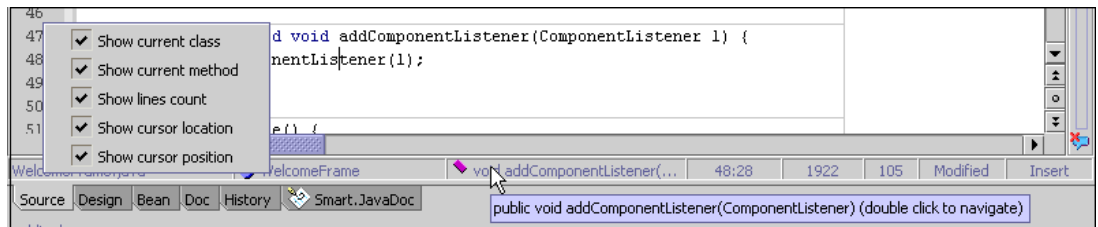


Figure 58 Advanced Text View Status Bar

The [Advanced Text View Status](#) provides all functionality of built-in JBuilder status bar and offers the following additional features:

- Displaying name of class corresponds to caret position (Java files only);
- Name of method corresponds to caret position (Java files only);
- Caret position (offset of caret location within editor document);
- Number of lines in editor document;
- Ability to instantly change read-only status (if file is not modified and is stored in file system) using mouse double click on file status label;
- Ability to change insert/override status using mouse double click on insert status label;
- Ability to invoke JBuilder “Go to line number” dialog using mouse double click on caret position or caret location label;
- Ability to invoke Browse.Insight popup using mouse double click on the class name label (if one is visible);

- Ability to invoke Browse.Members popup using mouse double click on the method name label (if one is visible);
- Ability to customize set of labels should be visible within Advanced Text View Status using popup menu which can be invoked using right mouse click on any status label;

### Smart.Braces

---

**Smart.Braces** is a tool that allows easy creation of matching braces right while you are typing. Just type an opening brace and **Smart.Braces** will automatically add the closing one. In addition to braces completion, **Smart.Braces** supports completion of string and character enclosing symbols - " and ' , respectively.

**Smart.Braces** adds closing characters after the opening ones for all characters except curly braces; the closing curly brace is inserted into the next line and may require an additional line, for the cursor with the appropriate indent to be placed (according to the *Complete curly brace and indent* option).

#### Options Dependency

---

You can control the behavior of **Smart.Braces** using the *Editor Options | Editor | Editor Options* tree view - expand the **Smart.Braces** options node and turn on or off options you need.

#### NOTE:

With a non-standard JBuilder keymap used (such as Vi/VIM), **Smart.Braces** may conflict with keymap settings. Apparently, for the VI keymap, ' and " symbols may be overridden by Smart.Braces. The reason of such behavior lies in the features of the vi implementation (not absolutely correct implementation of the Keymap default action).

However, you can disable the part of the **Smart.Braces** functionality, which leads to the conflict. To do this, please add the following lines to your JBuilder.config file (located in the JBuilder/bin directory):

```
vmparam -DProductivity.Smart.Braces.CompleteCharacters=no
```

```
vmparam -DProductivity.Smart.Braces.CompleteStrings=no
```

### Matching.Code.Highlight - *Pro!*

---

Java is well-structured language, but several statements exists that greatly decrease readability of the program – those ones, which breaks normal code flow execution.

In some cases, especially if source code is written by another developer, it can be hard to understand to which statement, for example, break one points too. The situation become even worse if these break statement has label and labeled statement is far from break one. The Matching.Code.Highlight become really invaluable in such situations.

Matching.Code.Highlight tool provides help in source code investigation and helps to understand which code is matching to one at the caret position. In addition, this tool provides easy navigation to it.

```

1479     for (int i = 0; i < anInnerClasses.length && result == null; i++)
1480     {
1481         JotClass current = anInnerClasses[i];
1482         if (current.getName().equals(aClassName))
1483         {
1484             result = current;
1485             break;
1486         }

```

Figure 59 Matching.Code.Highlight

It introduces the following features:

- Ability to highlight the break target statements for return, break, labeled break (break <label>), continue, and labeled continue (continue <label> statements);
- Ability to highlight matching code using dotted path with arrow on the Gutter;
- Ability to view matching brace code when it's has been scrolled out of view. The special popup window shows the matching code in the top of the editor;
- Ability to navigate to matching code using **Ctrl+Shift+\<** (CUA) shortcut.

### Options Dependency

You can control the behavior of [Matching.Code.Highlight](#) using the *Editor Options / Productivity! / Tools* property page. Using this page you are able to specify whether the code popup window should be displayed as well as to set the popup window invocation delay.

### Smart.Braces.Highlight - *Pro!*

The [Smart.Braces.Highlight](#) tool offers matching braces highlight and navigation operations.

```

for (int i = 0; i < anInnerClasses.length && result == null; i++)
{
    {
        result = current;
        break;
    }

    if (aClassName.startsWith(current.getName()))
        result = obtainClass(JotUtils.obtainInnerClasses({JotClassSource
}

```

Figure 60 Smart.Braces.Highlight with hint window

The major features of [Smart.Braces.Highlight](#) are:

- Ability to find and highlight matching brace when the caret is placed at any side of a brace;
- Ability to navigate to the matching brace using only one shortcut **Ctrl+Backslash** (CUA) for opening and closing ones;
- Ability to view matching brace code when it's has been scrolled out of view. The special popup window shows the open brace code in the top and close brace code in the bottom of the editor correspondingly. Please note that the close brace code can be shown for try/catch/final and do/while statements only;
- Ability to show matching braces scope on the Gutter.

#### Options Dependency

---

The [Smart.Braces.Highlight](#) behavior can be controlled using the *Editor Options / Productivity! / Tools* property page.

#### Changes.Highlight - *Pro!*

---

The [Changes.Highlight](#) tool highlights changed lines of code using the special marks in the gutter. The main features are:

- Change marks can track amount of changes in the particular line of code and reflect the number of them using different colors. Thus, more times line was edited, more bright color will be used to highlight changes;
- Full support of undo/redo functionality as well as re-reading of document;
- Full support of MVC architecture that allow tracking and showing changes in the document across different views and browsers.

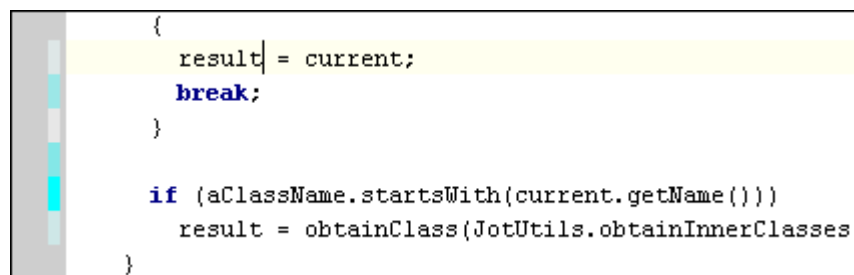


Figure 61 Changes.Highlight and Current Line Highlight


#### Current Line Highlight - *Pro!*

---

The [Current Line Highlight](#) increases usability of the JBuilder editor. As it follows from its name, this tool simply highlights the line of code at the caret position using specified background color.



Figure 62 Current line highlighting switch

Line highlighting can be turned on/off using the  button from the View Toolbar (one that is placed at the left bottom of the editor near horizontal scrollbar left corner).

## Classes and Methods Separator - *Pro!*

This is another tool included into Productivity! intended to increase readability of the Java source code. It visually separates classes and methods from each other by painting horizontal line at the top of corresponding classes and methods declarations.

Please note that this feature depends on the errors in Java source file. If there are some severe syntax errors the Java source code parser is unable to parse code correctly. In such a case some of the dividers may be displayed incorrectly.

```

jMenuFileExit.setText("Exit");
jMenuFileExit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
    }
});
jMenuHelpAbout.setText("About");
jMenuHelpAbout.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
        class LocalClass
        {
            public void a() {}
        }
    }
});
jMenuBar1.add(jMenuHelp);
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
    }
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
}
  
```

Figure 63 Visual separation of Methods and Classes

## IDE Improvements

---

Productivity! offers rich set of powerful JBuilder IDE improvements, intended to increase JBuilder usability.

These tools covers very common tasks and allows significantly minimize time required for their completion.

These tools help in solving of the following tasks:

- Synchronization of currently opened file with appropriate node in the JBuilder Project View – [Project View Synchronizer](#);
- Getting more detailed information about Java file structure as well as filtering of Java Structure content – [Advanced Java Structure](#);
- Synchronization of the current caret position in the editor with Java code with appropriate structure element – [Java Structure Synchronizer](#);
- Quick obtaining information about read-only status of file and changing it – [Change.ReadOnly](#);

### Project View Synchronizer - *Pro!*

---

The [Project View Synchronizer](#) provides ability to synchronize file is currently being editing with the corresponding node in the JBuilder Project View. There is possibility to have continuous synchronization that automatically tracks the changing of the current file and finds corresponding node in the Project View.



Figure 64 Force Project View Synchronizer Button

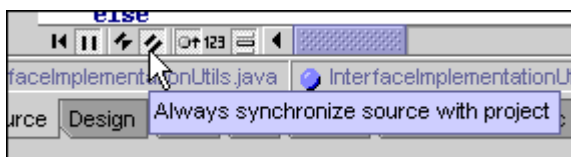




Figure 65 Project View Synchronizer Buttons

If continuous synchronization is disabled, there is an ability to force synchronization manually. [Project View Synchronizer](#) behavior can be controlled using  and  View Toolbar buttons placed at the left bottom of the editor (near horizontal scrollbar left corner).



## Options Dependency

The [Project View Synchronizer](#) behavior can be controlled using the *IDE Options / Productivity!* property page which may be used to specify if all nodes except the current one should be collapsed after [Project View Synchronizer](#) invocation.

## Structure View Synchronizer - *Pro!*

This tool allows synchronizing of Java structure element at the caret position in the editor with the corresponding element in the [Java Structure View](#). There is possibility to have continuous synchronization that automatically tracks caret position changes and finds corresponding element in the Structure View. If continuous synchronization is disabled, there is an ability to force synchronization manually.

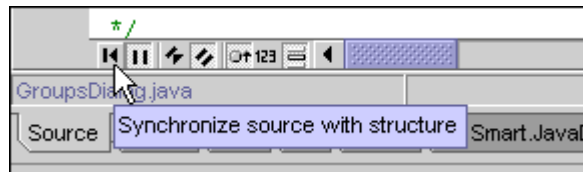


Figure 66 Java Structure Synchronizer Buttons

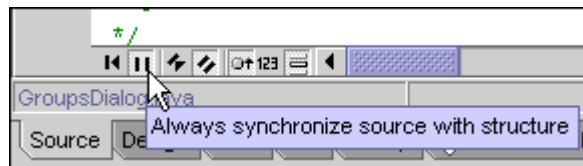


Figure 67 Java Structure Synchronizer Buttons

[Structure View Synchronizer](#) behavior can be controlled using the **⏪** and **⏸** View Toolbar buttons placed at the left bottom of the editor (near horizontal scrollbar left corner).

## Change.ReadOnly- *Pro!*

This tool allows easy viewing and managing read-only status for file nodes. The main features are:

- Ability to change read-only status (if file is stored in file system) for one or any files using popup menu on viewer tab, file node in the Project View or by mouse double click on appropriate panel in status bar (the last approach is applicable to Java files only);
- Ability to highlight read-only status using red dot in the top-right corner of the node icon (for Java files);
- Ability to highlight modified status using blue dot in the top-right corner of the node icon (for Java files);
- Ability to track outer changes of read-only status as well as file modifications .



Figure 68 Java Structure Synchronizer Buttons

## Navigation Tools

---

Productivity! offers rich set of powerful tools intended to simplify navigation and allow quick finding of required information.

There are several types of navigation are supported:

- Navigation to appropriate class using short class name- [Browse.Insight](#);
- Quick navigation to appropriate member of Java Class – [Browse.Members](#);
- Navigation to particular symbol definition – [Hyperlink.Navigate](#);
- Navigation to appropriate bookmark in editor – [Persistent.Bookmarks](#);
- Navigation and iteration to source code element (class, method, fields) code issue, to-do comment of search result – [View Navigator](#) and [Navigator](#);

### Browse.Insight

---

[Browse.Insight](#) allows quick finding Java classes with short names corresponding to the word at the cursor position, browsing them or opening the appropriate help topic for them.

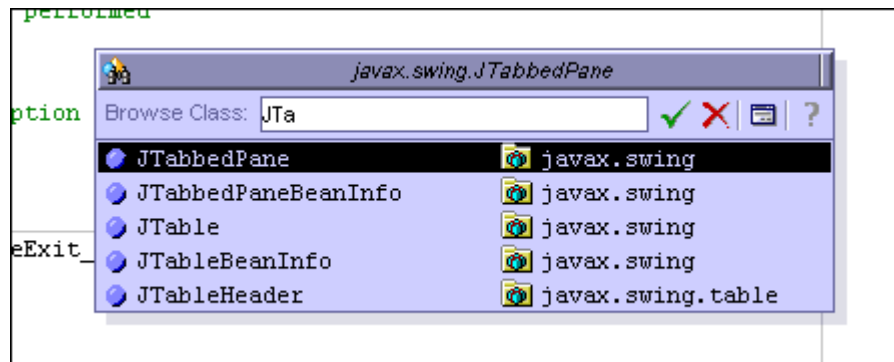


Figure 69 [Browse.Insight](#) Popup Window

To invoke [Browse.Insight](#) press **Ctrl+Minus** (CUA). The [Browse.Insight](#) popup will be shown with the list of classes matching the word at the cursor position. The list may be empty if there are no matching classes though. To find matches, type a word in the Browse Class edit box and [Browse.Insight](#) will dynamically rearrange the classes' list to show the matching ones.

You can select a class navigating through the list with the help of the usual keyboard. An Alternative to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

#### [Browse.Insight](#) Actions

---

Press **Enter** when you find the required class and [Browse.Insight](#) will open this class in the browser.

Press **Ctrl + Enter** when you find the required class and [Browse.Insight](#) will open the appropriate help topic for it.

Also, there is a possibility of employing the *Browse Classes dialog*, which shows packages structure and allows choosing a class by specifying its full path. You can use the appropriate button in the top left corner of the popup to invoke it.

### Options Dependency

Please note that the set of classes shown in [Browse.Insight](#) depends on *Packages Exclusion* settings on the *Project Properties | Productivity! | General* property page.

You can adjust the way of classes sorting as well as the algorithm used for classes search using the *Editor Options | Productivity! | General* property page. There, using the *Productivity! Insights Usage option*, you can specify whether [Productivity! Browse.Insight](#) tool or JBuilder built-in Browse classes should be invoked.

## Browse.Members

[Browse.Members](#) allows quick finding members belonging to the current discovered context and browsing them.

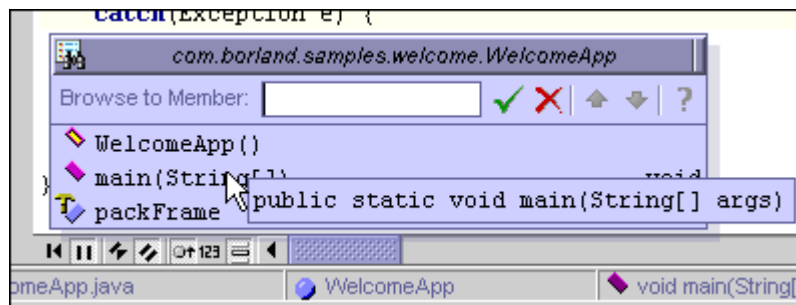


Figure 70 [Browse.Members](#) Popup Window

Press **Alt+Minus** (CUA) when editing a file to invoke [Browse.Members](#) Insight. The [Browse.Members](#) popup will be shown with the list of members (either classes, methods or fields) matching the word at the cursor position. The list may be empty if there are no matching methods though. To find matches, type the word in the Browse Member edit box and [Browse.Members](#) will dynamically rearrange the members' list to show the matching ones. You can also leave the [Browse.Members](#) edit box blank to view all the members for navigation purposes.

[Browse.Members](#) highlights the members with names exactly matching the typed word using bold font and abstract methods using italic font.

You can select a member by navigating through the list with the help of the usual keyboard. An Alternative way to do it is to continue typing the word; the list selection will be changed to produce the closest match possible.

Press **Enter** when selecting a member and [Browse.Members](#) will browse it.

## Hyperlink.Navigate

---

[Hyperlink.Navigate](#) is a tool allowing easy and convenient navigation with a method similar to that of the JBuilder built-in Symbol Insight tool.

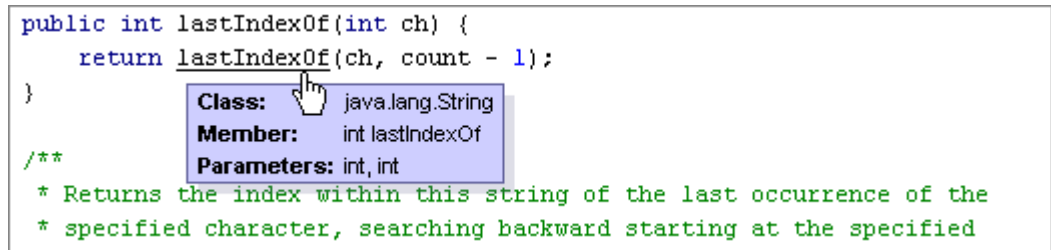


Figure 71 Hyperlink.Navigate with Hint that Describes Identifier Under Cursor

To invoke [Hyperlink.Navigate](#), press and hold the **Ctrl** key pointing the mouse over the identifier you are going to navigate to. A hyperlink will appear, and if you press the left mouse button, JBuilder will navigate to it (in the same manner as Symbol Insight does).

If you place the mouse over the identifier with the **Ctrl** key pressed, after some delay the [Hyperlink.Navigate](#) popup appears that contains information about the symbol under the cursor.

### Options Dependency

---

You can customize delays used for invocation and closing of the [Hyperlink.Navigate](#) popup window using the [Hyperlink.Navigate Delays options](#) on the *Editor Option / Productivity / Delays* property page. Also, you can specify whether [Hyperlink.Navigate](#) should be invoked during a debug session using the *Invoke insights during debugging* option on the *Editor Option / Productivity / Usage* property page.

## Search Results and References Highlight - *Pro!*

---

This tool is intended to highlight in the editor various things found during search or find references operations (those things should be listed in the message pane as well). To place highlights the special button on the view toolbar should be used. If nothing selected in the message pane this tool uses the most recent search results related to the current file. But it's possible to exactly specify result set to highlight by selecting root node in the search tree (e.g. "Direct usages" or "Declarations"). To navigate through the highlights usual Navigator keys (Ctrl+Page Up/Page Down (CUA)) are used.

```

public WelcomeApp() {
    WelcomeFrame frame = new WelcomeFrame();
    //Pack frames that have useful preferred s
    //Validate frames that have preset sizes
    if (packFrame)
        frame.pack();
    else
        frame.validate();
}

```

Figure 72 Search Results and References Highlight

## Persistent Bookmarks - *Pro!*

---

JBuilder provides bookmark functionality that allows to set bookmark associated with particular line of the file and the return to it. However, JBuilder built-in implementation of the editor bookmarks has some drawback – all bookmarks are actually associated with editors, not with files thus bookmarks are not persistent between JBuilder sessions. The [Persistent.Bookmarks](#) tool included into Productivity! is free from drawbacks mentioned above and introduces advanced bookmarks concept as well as offers lots of new possibilities.

Each bookmark is linked to a project (if any) and to a file and is hold its own location (as line number) in the file as well. Every bookmark can optionally contain the *Description* attribute those can be specified to allow easy identifying of particular bookmark.

There are two types of bookmarks are currently supported – persistent ones and temporary ones. All bookmarks with *Persistent* attribute enabled are stored to the IDE properties on JBuilder exit and, correspondently, all such bookmarks are loaded during the following JBuilder run. This behavior allows pointing to the most used files and easy navigating them anytime.

The following actions can be executed during navigating to a persistent bookmark:

1. If particular bookmark contains reference to a project this project will be opened (if it still not opened yet) and activated;
2. The file which bookmark refers to will be opened (if it still not) and activated;
3. If bookmark location within file has a valid value, the editor will be scrolled to the bookmark' location.

Persistent bookmarks can be toggled and navigated using usual JBuilder shortcuts. Numbered bookmarks are supported too. The only way to navigate unnumbered bookmark is using the [Persistent.Bookmarks.Navigate](#) tool.

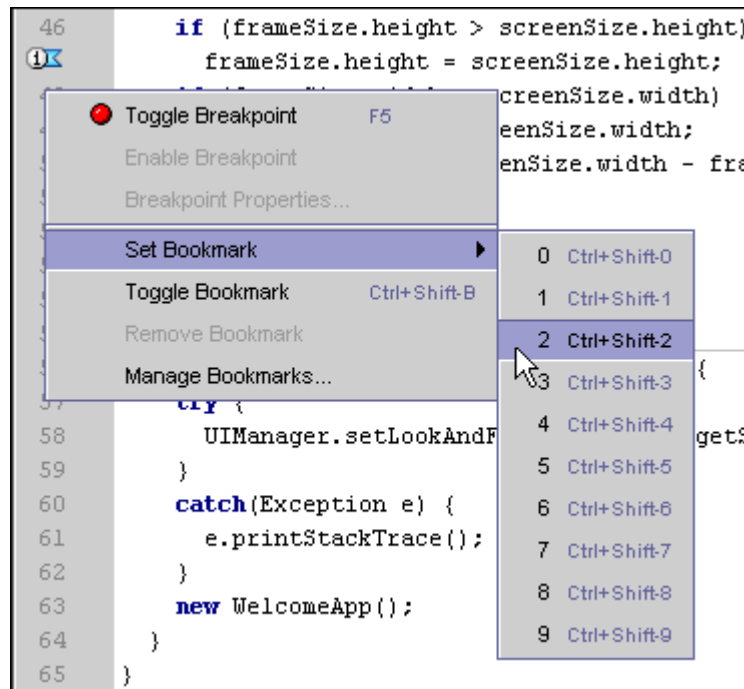


Figure 73 Setting a Bookmark Using the Gutter Context Menu

Another way of working with persistent bookmarks is using context menu on the editor gutter.

Also, if bookmark has description associated with it, it may be displayed as hint if mouse is placed over bookmark icon.

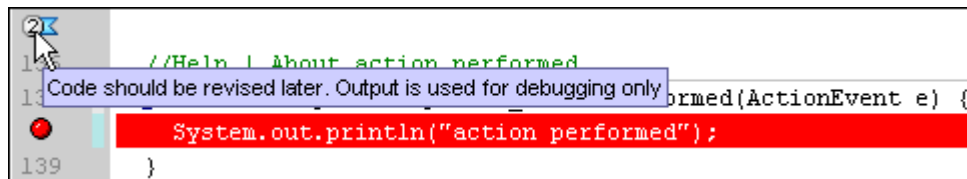


Figure 74 Hint with Description of Bookmark

**Persistent.Bookmarks.Navigate**

This tool allows easy navigation to any bookmark from the bookmark list. [Persistent.Bookmarks.Navigate](#) can be invoked using the **Alt+Shift+B** (CUA) shortcut.

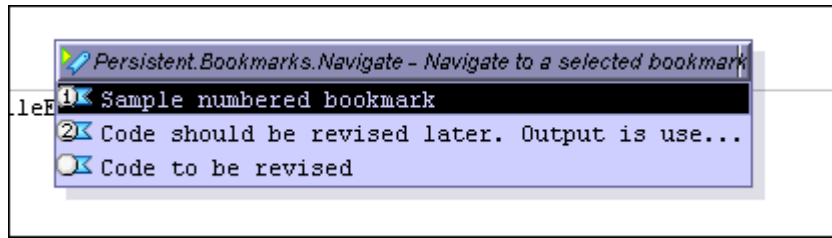


Figure 75 Persistent.Bookmarks.Navigate

### Manage Bookmarks Dialog

Productivity! provides rich functionality that allows bookmarks managing. The Manage Bookmarks Dialog allows maintaining the bookmarks list.

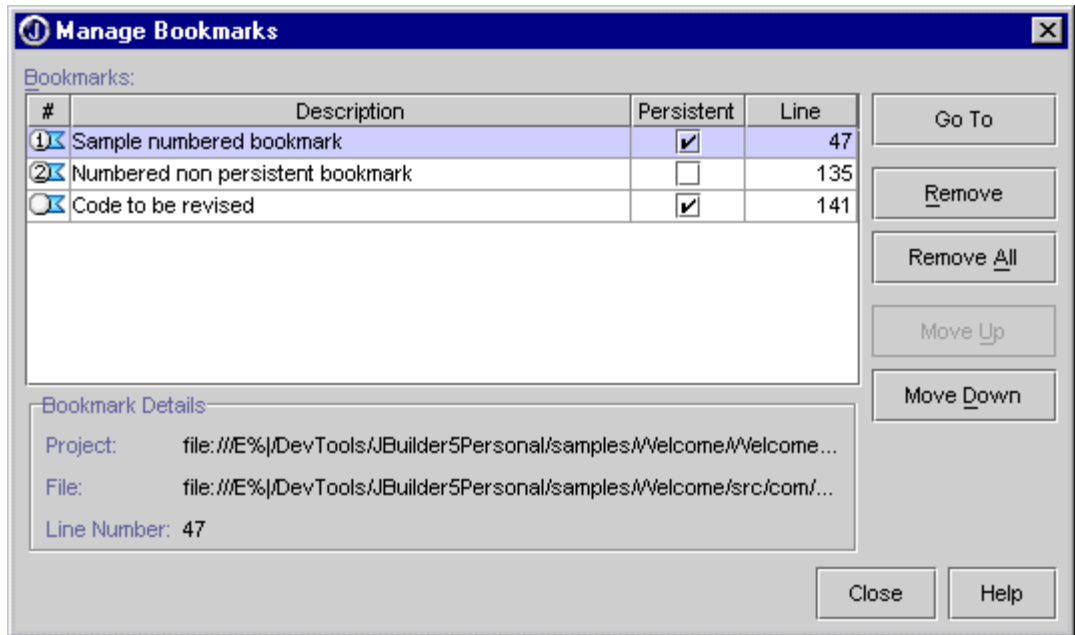


Figure 76 Manage Bookmarks Dialog

The *Manage Bookmarks* dialog can be invoked using the *Edit / Manage Bookmarks* menu item or using the editor gutter context menu.

This dialog provides the following controls, which allow viewing and maintaining the bookmarks list:

#### Bookmarks Table

The *Bookmarks Table* shows the list of bookmarks. Each bookmark occupies one row in the table while each table's column represents particular attribute of bookmark. The *Description* and *Persistent* columns are editable. This allows specifying values of them using in-place editing capabilities. If *Description* attribute is specified, it will be displayed in the [Persistent.Bookmarks.Navigate](#) tool and will be shown as tool tip when mouse is placed over bookmark in the gutter. If bookmark's *Persistent* attribute is turned off, such a bookmark will not be stored on JBuilder exit.



*Go To*

The *Go To* button provides navigation to the currently selected bookmark without closing the dialog.

*Remove*

The *Remove* button allows removing of the bookmark is currently selected.

*Remove All*

The *Remove All* button removes all bookmarks.

*Move Up*

The *Move Up* button allows moving currently selected bookmark up in the bookmarks list.

*Move Down*

The *Move Down* button allows moving currently selected bookmark down in the bookmarks list.

**Note:** The *Manage Bookmarks Dialog* allows actual maintenance of bookmarks list so no changes can be discarded or undone by closing this dialog.

### View Navigator and Navigator.Insight - *Pro!*

---

The [View Navigator](#) tool provides ability to control set of source code related navigation operations using the same way and uniform manner. The main idea of this tool is utilizing of pair of shortcuts and a pair of appropriate buttons to execute 'previous' and 'next' navigation operations against specified Navigate Object. Following Navigate Objects are currently supported for Java files:

- Warning or Error
- Low Priority Issue
- Any Issue
- Editing Point
- Method
- Class
- Class or Method
- Search Result
- Local Reference

Following Navigate Objects are available for other types of text files:

- Editing Point
- Search Result

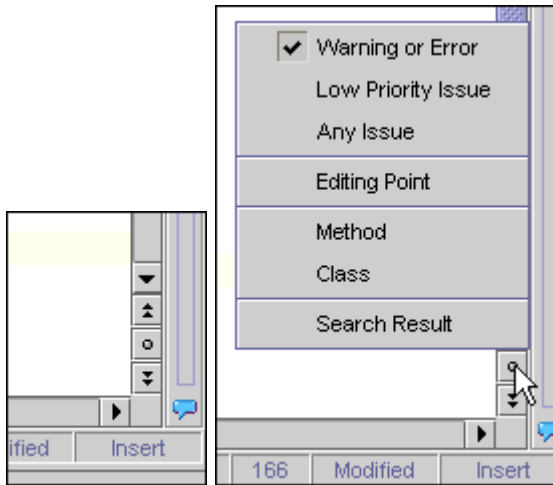


Figure 77 View Navigator and View Navigator menu

The Navigator control is placed at the bottom right of the editor view (right under the vertical scrollbar) and includes the *Previous*, *Next* and *Select Object To Navigate* buttons. The *Previous* and *Next* buttons allows navigating through the objects (you can use **Ctrl+Page Up** and **Ctrl+Page Down** shortcuts (CUA) too). The *Select Object To Navigate* button allows choosing an object to navigate by using appropriate menu items from popup menu shown on button release. Another way choose object type to navigate is using [Navigator.Insight](#), which can be invoked using the **Alt+Shift+N** (CUA) shortcut.

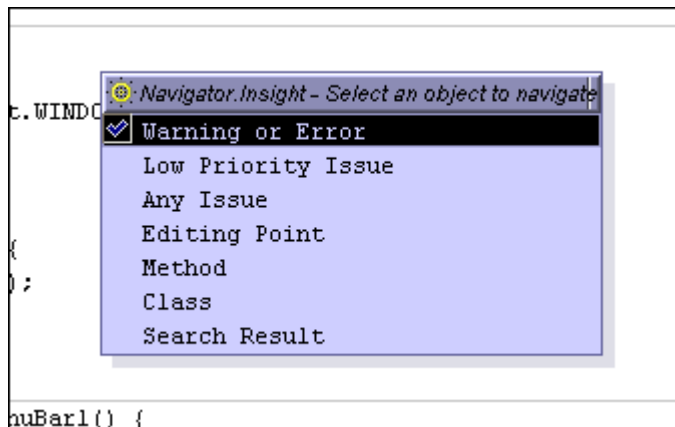


Figure 78 Navigator.Insight

## Information Tools

Productivity! includes several advanced tools those allows you instantly get information about required Java source code element.

With Productivity! Information tools you can:

- Easily view help topic (if one exist) for the identifier in the caret position – [Help.Insight](#) and [Hyperlink.Help](#);
- Get instant help selected item in JBuilder Code Insight – [Help.Insight](#);
- Get full information about context of source code in the current caret position – [Context.Insight](#);

### Help.Insight

[Help.Insight](#) is a tool allowing you to easily view help topics, if any, for an identifier or a member within the current context in the cursor position. Help is shown in a convenient popup window. [Help.Insight](#) extracts and displays information relating to a particular code only - for example, for a particular method, not for all classes.

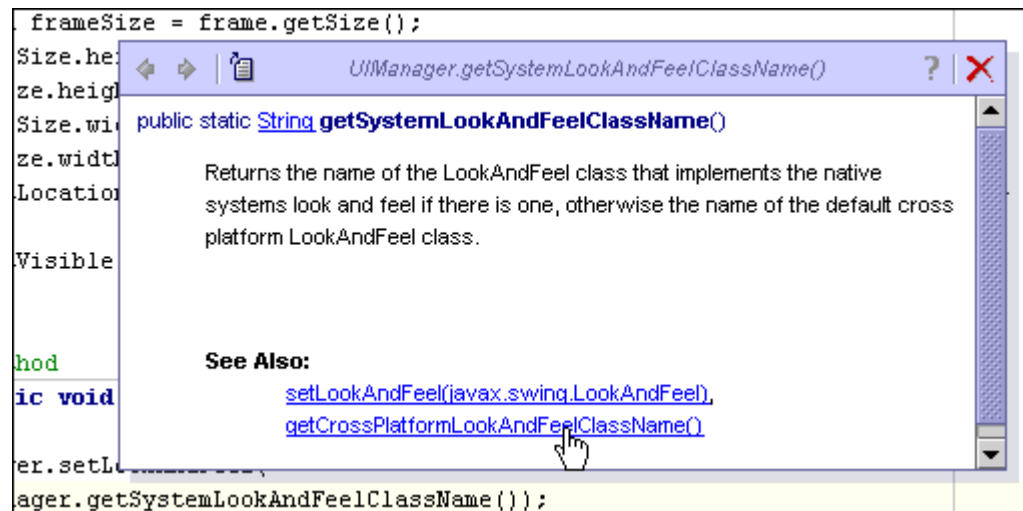


Figure 79 Help.Insight Popup Window

To invoke [Help.Insight](#) for a symbol at the cursor position, place the cursor over the identifier for which you need help and press **Shift+F1** (CUA).

To invoke [Help.Insight](#) for a member within the current context, place the cursor in the bounds of a method or class for which you need help and press the **Alt+F1** (CUA) shortcut. [Help.Insight](#) shows the appropriate JavaDoc help topic, if any, or tries to find and show the appropriate one for a super class or method in other case.

### Navigation Pane

The Navigation Pane at the top of the popup shows different gadgets intended to control the popup. You can use the Back and Forward buttons (or **Alt+Left** and **Alt+Right** keys, respectively) to navigate through the help topics history or you can use the Open the

Whole Topic button to open the complete help topic in the Help Viewer window. The Context label shows the context or an HTML file name depending on the ability to resolve any.

### Hyperlink.Help

An Alternative way to invoke [Help.Insight](#) is to use [Hyperlink.Help](#) by placing the mouse cursor over an identifier for which you need help while holding the **Alt** button.

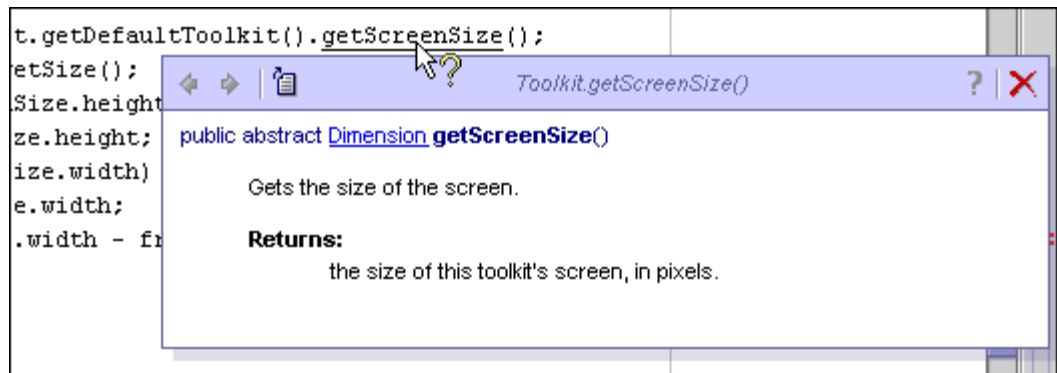


Figure 80 Help.Insight Popup Invoked via Hyperlink.Help

In this case, for space-saving purposes, the [Help.Insight](#) popup doesn't show the Navigation Panel. It will only be shown after activating any hyperlink within the popup window.

### Integration with Other Insights

Another feature of [Help.Insight](#) is integration with JBuilder built-in MemberInsight and other Productivity! Insights.

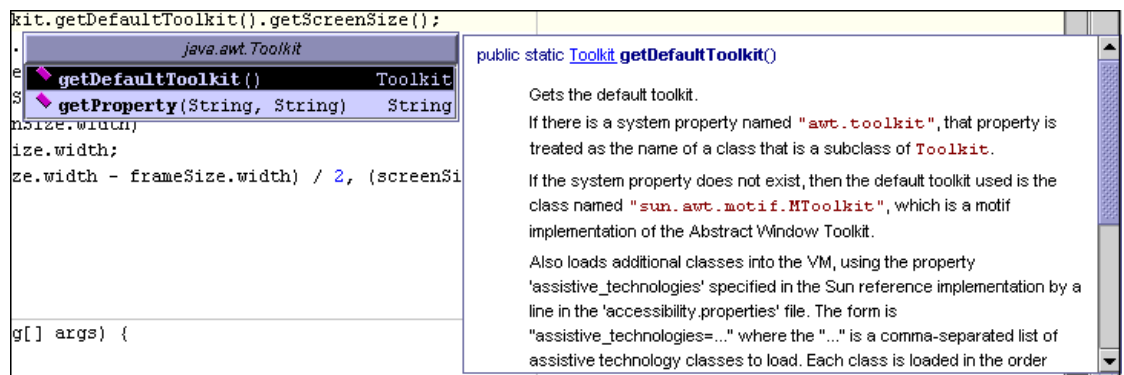


Figure 81 JBuilder Member Insight with Help.Insight Invoked

If such integration is enabled, the popup window of [Help.Insight](#) will be automatically shown for a currently selected item in Member Insight or Productivity! Insight popup. You can also force showing [Help.Insight](#) for a selected item using **Shift+F1** shortcut (CUA).

## Options Dependency

---

You may enable integration of [Help.Insight](#) with other Insights using *Help.Insight* options on the *Editor Options | Productivity! | Usage* property page. To specify delays of [Help.Insight](#) invocation you can use the *Editor Options | Productivity! | Delays* property page.

## Hyperlink.Help

---

[Hyperlink.Help](#) is a tool that allows easy and convenient viewing help topics for particular symbols.

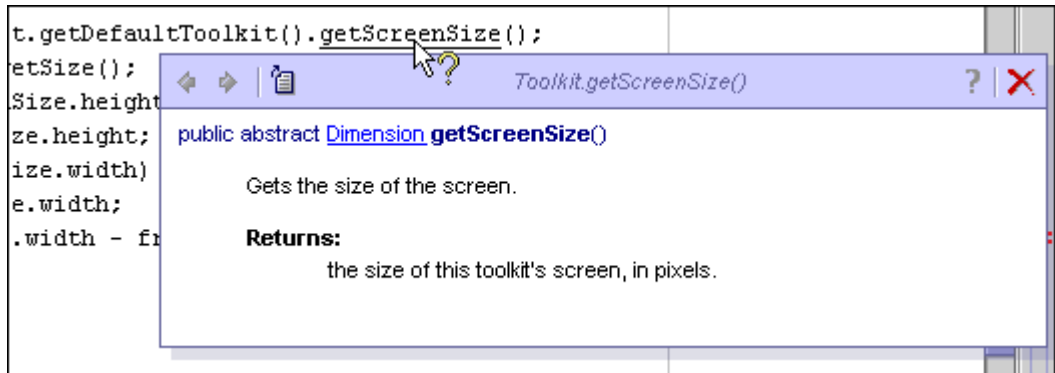


Figure 82 Hyperlink.Help Popup Window with Help for Method

To invoke [Hyperlink.Help](#), press and hold the **Alt** key and point the mouse over the identifier, which you need help with. The identifier becomes a hyperlink, and if you press the left mouse button, the built-in JBuilder help is shown for it.

If you place the mouse over the identifier with the **Alt** key pressed, after some delay the [Help.Insight](#) popup appears that contains exact help about the symbol under the cursor.

## Options Dependency

---

You can customize delays on invocation and closing of the [Hyperlink.Help](#) popup window using the [Help.Insight](#) options on the *Editor Option | Productivity! | Delays* property page. Also, you can specify whether [Hyperlink.Help](#) should be invoked during a debug session using the Invoke insights during debugging option on the *Editor Option | Productivity | Usage* property page.

## Context.Insight

---

[Context.Insight](#) is a tool that allows you to check context of the current cursor position. [Context.Insight](#) collects information about all classes and methods and shows it using the Insight popup window.

To invoke [Context.Insight](#) please use the **Ctrl+Q** (CUA) shortcut.

```

Runnable runnable = new Runnable()
{
    public void run()
    {
        public class ContextUtils
        {
            public static class ContextUtils.PathDeclaration
            {
                void addPath(List aPath)
            }
            class ContextUtils$1 extends Runnable
            {
                public void run()
            }
        }
    }
}

```

Figure 83 Context.Insight Popup Window

Another feature supported by [Context.Insight](#) is navigation. If you place the mouse cursor over an identifier within the [Context.Insight](#) popup window, the hyperlink appears and activation of hyperlink leads to navigation to the identifier.

In some cases (particularly, within classes with significant amount of inner classes), the [Context.Insight](#) may display only upper class information. Such behavior is explained by definite limitations of the JBuilder JOT subsystem that requires significant amount of time (up to tens of seconds) to retrieve information about the inner classes. To avoid hang-up of JBuilder, the time required for context information gathering may be limited in *Editor Options | Productivity! | Delays* page. So, if JOT provides no data within this interval, only upper class information is provided. The same reason may cause relatively slow performance of [Context.Insight](#) if the cursor is on the white space between class methods. The same limitations may affect other tools using the same functionality (Override.Insight, Implement.Insight).

### Options Dependency

---

You can specify Context Discovering Timeout using the *Editor Options | Productivity! | Delays* property page.

## Productivity! Options

---

Productivity! Pro offers rich abilities for customizing its functionality and tuning it exactly for your own unique code style and your particular needs.

You can manage Productivity! settings in convenient and customary ways using standard JBuilder approaches for configuration.

Most of Productivity! settings are concentrated in two JBuilder dialogs: the Project Properties and Editor Options dialogs.

The *Project Properties* dialog contains the following property pages added by Productivity!:

- *General* where you can specify options for [Imports.Beautify](#) and Packages Exclusion.
- *Code Style* where you can specify options for Code Generation.
- *JavaDoc* where you can specify options for JavaDoc generation.
- *Cache* where you can specify options for managing Productivity! classes cache.
- *Assistant* where you can specify options for managing Assistant behavior.
- *Tools* where you can specify options for various Productivity! tools.

The *Editor Options* dialog includes the following property pages added by Productivity!:

- *General* where you can specify options for *Import Statements Generation*, *Search Options*, *Sorting options*, *Autocomplete*, *Insight Usage* and *Invocation insights during debugging*.
- *Usage* where you can specify how to use (or not use) the appropriate tools
- *Delays* where you can specify options for [Hyperlink.Help](#) and [Hyperlink.Navigate](#) invocation and closing delays, [Help.Insight](#) delay used for integration with JBuilder Member Insight, and *Context Discovering* timeout.
- *Smart.Templates* where you can customize the Smart.Templates behavior and maintain the templates list.
- *Assistant* where you can customize the Assistant behavior.
- *Tools* where you can customize miscellaneous tools behavior.
- *Smart.JavaDoc* where you can customize the Smart.JavaDoc behavior.

In addition, with the help of *Editor Options* Dialog you are able to customize options for the [Smart.Braces](#) tool. These options can be found on the Editor property page in the *Editor Options* tree view.

Also, the *IDE Options* dialog includes a property page added by Productivity!, which allows user to select the Metal theme to be used by JBuilder.

## Project Properties Dialog

---

The *Project Properties* dialog contains the following property pages added by Productivity!:

- *General* where you can specify options for *Imports.Beautify and Packages Exclusion*
- *Code Style* where you can specify options for Code Generation
- *JavaDoc* where you can specify options for JavaDoc generation
- *Cache* where you can specify options for managing Productivity! classes cache

### General Page

---

The Options page of the Productivity! Project Properties pages allows to specify the following options:

1. *Packages Exclusion*

To set these options for all new projects, choose *Project / Default Project Properties*.

### Packages Exclusion

---

Productivity! uses cache of classes included into particular project. By default, this cache includes all classes found according to JBuilder paths settings. In general, this includes: classes from JDK, classes in project libraries and project classes themselves. It is obvious that in large projects the amount of such classes may achieve several thousands. The Productivity! popup windows allow reducing the excessive amount of classes by eliminating those not used (such as `sun.*` and `sunw.*`, which are included into JDK but hardly used in your project directly) - with the help of this option you can exclude unnecessary packages.

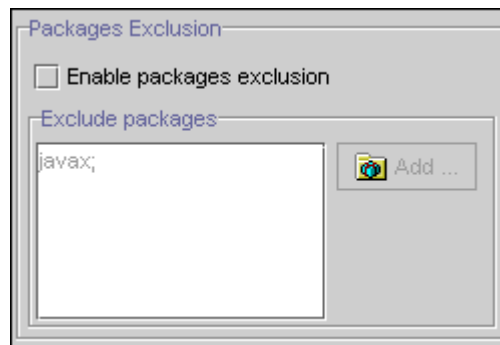


Figure 84 Package Exclusion Options

#### *Enable packages exclusion*

Select this checkbox if you wish to exclude classes belonging to particular packages from showing them in the Productivity! insights. With this checkbox disabled, all classes from Productivity! cache are shown.



*Exclude packages*

Using this panel, you can specify packages to be excluded. Either type the package name to exclude it manually (only valid Java symbols are allowed, ';' separates packages) or add the package using the *Select Package* dialog invoked by the *Add...* button. The sequence order of packages being specified is inessential.

**Code Style Page**

---

The Options page of the Productivity! Project Properties allows user to specify the following options:

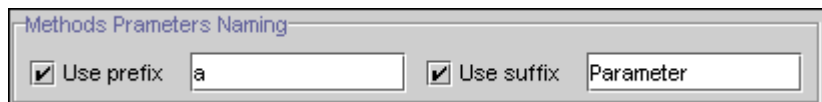
1. *Methods Parameters Naming*
2. *Fields Naming*
3. *Generate Throwing java.lang.UnsupportedOperationException Exception*
4. *General Options*

To set these options for all new projects, choose *Project | Default Project Properties*.

This property page allows customizing the code generated with Productivity! tools and adjust it to your personal coding style.

**Methods Parameters Naming**

This option allows customizing names of parameters used in methods generated by Productivity! tools. Any parameter name has a customizable prefix and suffix. With the appropriate checkbox enabled, you'll be able to specify the respective part of a parameter name in the edit box. In other words, you can specify the value to be used when naming parameters.



**Figure 85 Methods Parameters Naming Options**

Productivity! tools generate names of parameters if their actual names are unknown (when source code of a class is unavailable). By default, it utilizes usual Java convention for parameters naming, but you can force it to use prefixes and/or suffixes according to your own requirements.

**Fields Naming**

This option allows customizing names of fields according to the coding style you prefer.

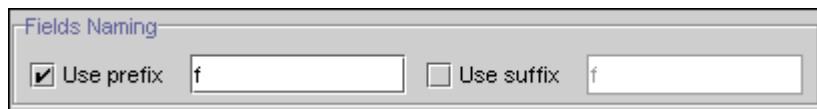


Figure 86 Fields Naming Options

You can customize a prefix and suffix of a field name. With appropriate checkbox enabled, you'll be able to specify the respective part of a field name in the edit box.

The current version of Productivity! uses this option in `GetSet.Creator`. Depending on values specified, `GetSet.Creator` can define the appropriate name of the get/set method (by removing a prefix and suffix) and the appropriate parameter names.

**Generate Throwing ...**

Using these radiobuttons you can exactly specify the rules of code generating in the method body. If you enable *Generate always* radiobutton, Productivity! always generates method body with TODO comment and the code that throws `java.lang.UnsupportedOperationException` exception.

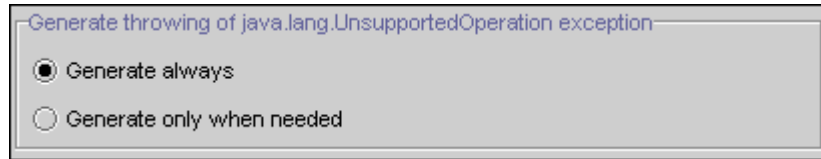


Figure 87 Generate throwing of UnsupportedOperationException Options

If you enable *Generate only when needed* radiobutton, Productivity! generates the code that throws exception only for the method with non-void return type.

**General**

These options provide more opportunities for you to fine-tune the code generated by Productivity!

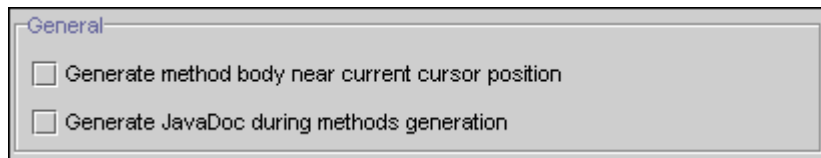


Figure 88 General Code Style Options

*Generate method body near the current cursor position*

This option allows you to specify the anchor position where the generated code will be inserted. If the appropriate checkbox is selected, the whole code will be generated in the position close to cursor (if the cursor is within a method, the code will be generated near this method). If this option is disabled, all the methods will be inserted in the end of a class definition and the constructors will be inserted after the last defined constructor. The only exception is the generation of get/set methods - if this option is disabled, a get/set method will be inserted after the appropriate set/get method.

*Generate JavaDoc during methods generation*

This checkbox is used to specify whether the JavaDoc comment templates will be generated during the methods generation. If you enable it, all the methods generated by the Productivity! tools will include the JavaDoc comment templates (the same as those produced by Easy.JavaDoc). The only exception is the generation of anonymous inner classes - JavaDoc will never be generated during the anonymous inner class generation.

**JavaDoc Page**

---

The [Easy.JavaDoc](#) page of the Productivity! *Project Properties* pages provides the following options:

1. *Policy for handling the existing JavaDoc comments*
2. *Methods JavaDoc Generation*
3. *Classes JavaDoc Generation*
4. *Auto Generation*

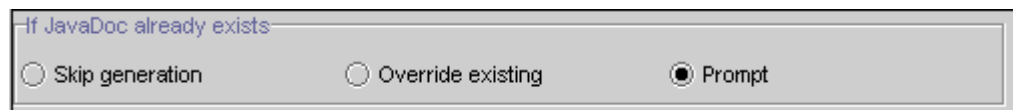
To set these options for all the new projects, select *Project / Default* from the *Project Properties*.

All the options on this page are applicable to JavaDoc generation by both manual invocation of [Easy.JavaDoc](#) (default shortcut is **CTRL+D**) and by invoking [Easy.JavaDoc](#) during method generation (in *Override.Insight*, *Constructor.Insight*, [Implement.Insight](#) and [Smart.Instantiate](#) tools).

Please note that all these options are not applicable to the code generated for anonymous classes, since JavaDoc is never generated for them.

**If JavaDoc  
Already  
Exists**

This option allows you to specify the processing policy for the existing JavaDoc comments.



**Figure 89 If JavaDoc exists Options**

You may define how [Easy.JavaDoc](#) will handle the existing JavaDoc comments. If JavaDoc already exists for a method or class, [Easy.JavaDoc](#) may either skip the generation of JavaDoc template and override the existing block by its own one, or prompt your confirmation for overriding of the existing comment.

**Methods  
JavaDoc  
Generation**

These options allow you to specify the tags that will be included into the generated JavaDoc template for the method.



Figure 90 Methods JavaDoc Generation Options

By default, [Easy.JavaDoc](#) always generates `@param`, `@throw` and `@return` (except void methods and constructors) tags based on the method definition. However, you may expand the content of generated template using Methods JavaDoc Generation option. You may select the appropriate checkbox to enable generation of the corresponding tag. Please note that if you select the "Generate `@author`" check box, [Easy.JavaDoc](#) will use the name of the Author as specified on the *Project Properties | General* property page.

---

#### Classes JavaDoc Generation

This option allows specifying the tags that will be included into the generated JavaDoc template for the method.

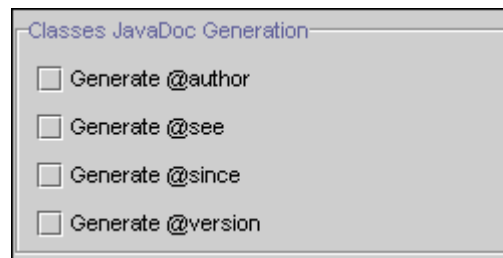


Figure 91 Classes JavaDoc Generation Options

By default, [Easy.JavaDoc](#) always generates a description only. However, you may expand the content of generated template via the Methods JavaDoc Generation option. You may select the appropriate checkbox to enable generation of the corresponding tag. Please note that if you select the *Generate @author* checkbox, the [Easy.JavaDoc](#) will use the name of the Author as specified on the *Project Properties | General* property page. The same happens with the *Generate @version* checkbox.

---

#### Auto Generation

You may specify whether default comments should be generated for the get/set methods using Automatically generate text of comments for get/set methods. Please note that this option is applicable only to [GetSet.Creator](#) tool.

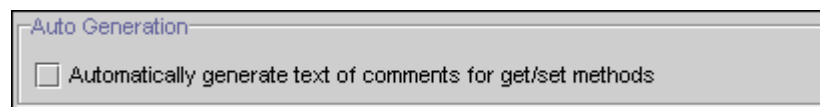


Figure 92 JavaDoc Auto Generation Options

If you enable this checkbox together with the automatic generation of JavaDoc during methods generation (*Project Properties | Productivity! | Code Style | General*), then the [GetSet.Creator](#) tool will insert default description for the method, default description for the method return value (getter) and default description for the method parameters (setter).

### Cache Page

---

The Cache page of the Productivity! Project Properties pages provides the following options:

1. *Autorefresh option*
2. *Refresh groups*
3. *Refresh Now*

To set these options for all the new projects, select *Project | Default* from the *Project Properties*.

The main goal of Productivity! is to increase the developers' productivity to its maximum. Since, presumably the application will be frequently used, it should work as quickly as possible. The project may contain several thousands of classes (including the classes directly included into the project, JDKs and required libraries classes) and constant search through them would be highly inconvenient. Thus, Productivity! builds classes cache right after the first invocation and then stores it to hard drive providing for future re-use. After cache build or load, Productivity! uses it for quick access to the classes according to the specified criteria.

Options grouped on this page allow you to control the process of class cache building and refreshing.

### Autorefresh Options

---

This option allows you to specify whether cache will be refreshed automatically.

The most frequently changed classes from all the classes used by the project are those included into the project itself, in other words, the classes developed by you within a project. Whereas changing JDK and adding or removing libraries are very rare operations, new classes within a project appear, change their location or become renamed every day.

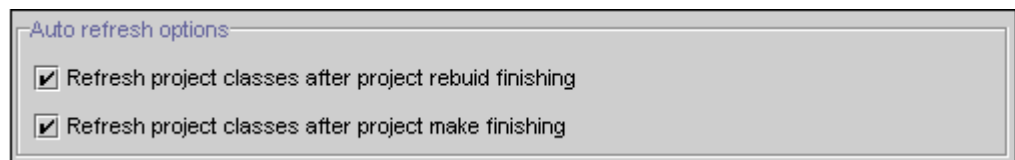


Figure 93 Cache Auto Refresh Options

The *Auto refresh options* are designed to make the cache content as up-to-date as possible, offering convenient usage of such Productivity! tools as [Class.Insight](#) and

Browse.Insight with your classes, and also to reduce the necessity of manual refresh of Productivity! classes cache.

This option enables your class cache to be refreshed after every successful project build or make. Refresh of classes included in the project is normally a short operation that requires much less time compared to project build, so we recommend that these options be always enabled.

Please note that the class cache refresh will be performed only under the condition that the whole project build or make is successfully completed (not just some of its files), and that there were no compiler errors during the build process.

**Refresh groups**

You may tune the cache refresh process by using the refresh groups. Refresh group is a set of packages that may be refreshed independently. Thus you may specify a set of refresh groups that would include the most frequently changing classes, and refresh their cache individually.

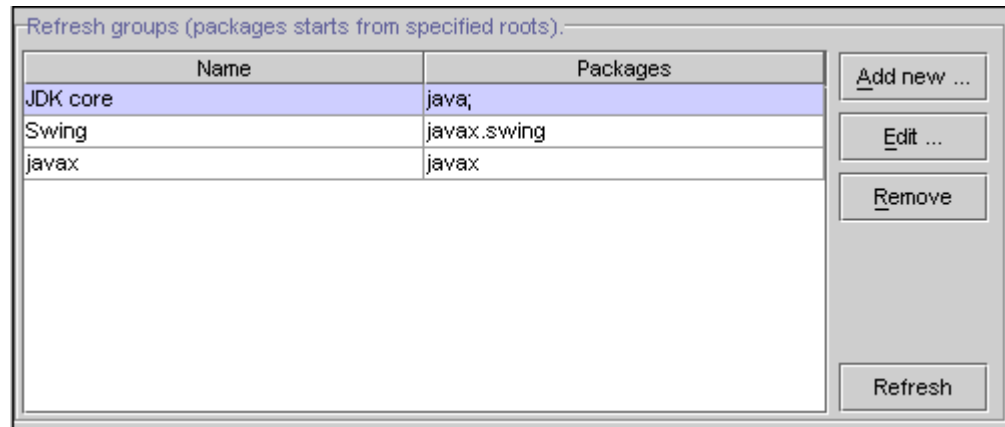


Figure 94 Refresh Groups Options

You may specify your refresh groups using the table shown above. Use *Add new...* button to create a new group, *Edit* button to edit the existing group, and *Remove* button to delete a group.

*Refresh* button allows you to refresh the selected group.

Please note that double-clicking a group row brings you up to group editing (similar to pressing the *Edit...* button).

Creating new groups as well as editing the existing ones is performed via the *New/Edit Refresh Group* Dialog.

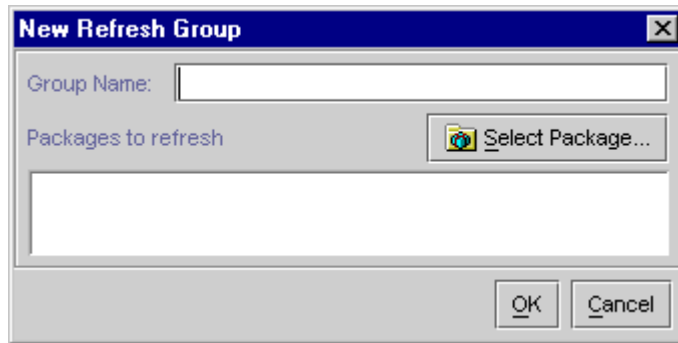


Figure 95 New/Edit Refresh Group Dialog

Please use the *Group Name* field to specify the name of the group to be refreshed, and the *Packages to refresh* field to specify the packages to be included into the group. You may specify the packages for inclusion by either manual typing (only valid Java symbols are allowed, separator for packages is ;), or by adding them via the *Select Package* dialog, invoked by the *Add...* button. The order of specified packages is not essential.

---

### Refresh Now

These options enable immediate start of the refresh operation for the Productivity! classes cache.

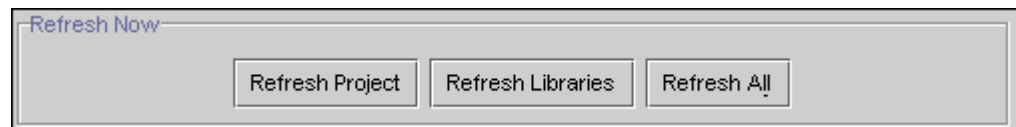


Figure 96 Refresh Now Options

You may choose from the following refresh types: of the classes included into the project only; of the project libraries (which is crucial to do after adding or removing libraries); or of the whole cache (including the classes from JDK).

### Assistant Page

---

The Assistant page allows specifying the following options:

1. *Auto Import Policy*
2. *Frequently Used Classes*

To set these options for all the new projects, select *Project | Default* from the *Project Properties*.

---

### Auto Import Policy

This option panel allows maintaining the list of Auto Import Policy entries. Such list is used to determine, which action should be executed for each particular short class name during auto import process.

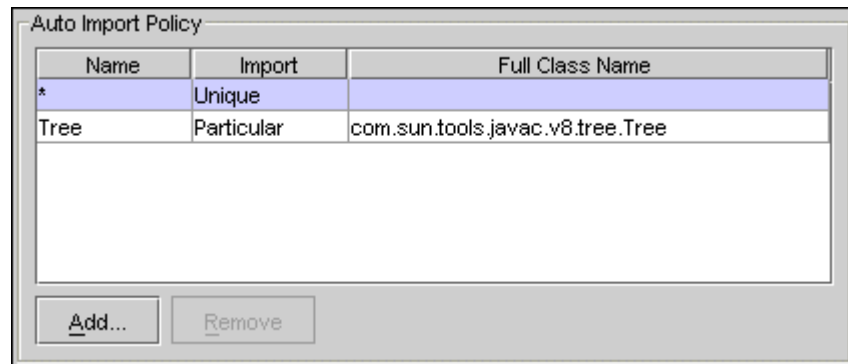


Figure 97 Auto Import Policy

Each entry occupies one row in the list and its properties are shown in the appropriate columns. All those properties can be edited right in the list using "in-place editing" approach. Auto Import Policy entry properties are:

- Name - specifies short name of the class. Predefined entry with name "\*" allows specifying default behavior for all names not stated in the list.
- Import - specifies action to be done. Possible actions are:
  - Never - no class will be imported.
  - Any - first suitable class will be imported.
  - Particular - class with exactly specified name will be imported.
  - Unique - unique suitable class will be imported.
- Full Class Name - specifies full class name to import. This property can be applied to Particular action only.

The Add and Remove buttons allow adding new and removing existing entries. Note, predefined entry "\*" can't be removed.

---

#### Frequently Used Classes

This option panel allows maintaining the list of frequently used classes. Such list is used to determine particular class to import or prompt in case of existence of several different candidates.



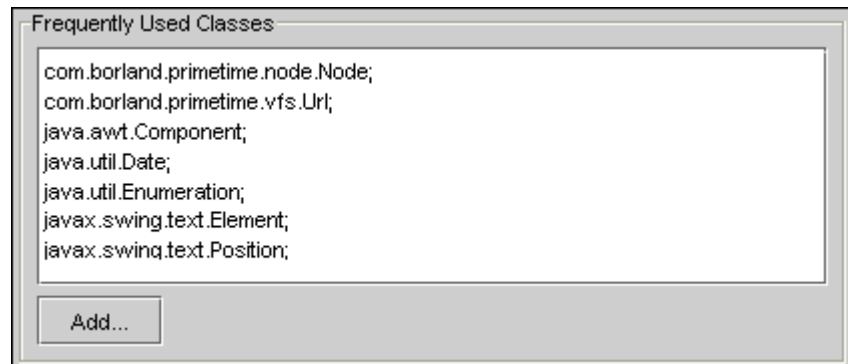


Figure 98 Frequently Used Classes

Each entry in that panel should be ended by ';' symbol. The sequence order of classes being specified is inessential. The Add... button allows adding a class using the Select Class dialog.

### Tools Page

---

The Tools page allows specifying the following options:

1. *Classes Highlight*

To set these options for all the new projects, select Project | Default from the Project Properties.

### Classes Highlight

---

Productivity! allows highlighting of certain classes using *Java / Extra keyword* style and these options allows fine tuning this functionality.

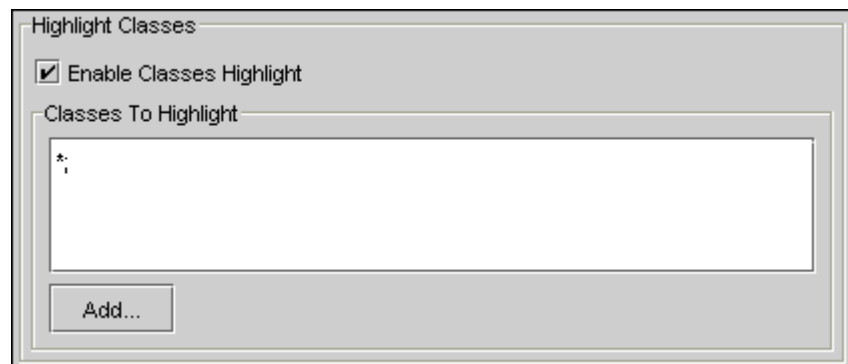


Figure 99 Classes Highlight

#### *Enable Classes Highlight*

Select this checkbox if you wish to turn highlight classes on.

#### *Classes to Highlight*

Using this panel, you can specify list of classes to be highlighted. Each entry in that panel should be ended by ';' symbol. The sequence order of packages and classes being specified is inessential. It's possible to use the following entries:

- \* - to highlight all classes.
- package.\* - to highlight all classes belonging to specified package.
- package.class - to highlight exactly specified class.

The Add... button allows adding a package or class using the Select Package or Class dialog.

## Editor Options Dialog

---

The *Editor Options* dialog includes the following property pages added by Productivity!:

- *General* where you can specify options for *Import Statements Generation*, *Search Options*, *Sorting options*, *Autocomplete*, *Insight Usage* and *Invocation insights during debugging*.
- *Usage* where you can specify how to use (or not use) the appropriate tools
- *Delays* where you can specify options for [Hyperlink.Help](#) and [Hyperlink.Navigate](#) invocation and closing delays, [Help.Insight](#) delay used for integration with JBuilder Member Insight, and *Context Discovering* timeout.
- *Tools* - **Pro!**
- *Assistant* – **Pro!**
- *Smart.Templates* – **Pro!**
- *Smart.JavaDoc* – **Pro!**

In addition, with the help of *Editor Options* Dialog you are able to customize options for the [Smart.Braces](#) tool. These options can be found on the Editor property page in the *Editor Options* tree view.

### Smart.Braces Options (Editor Options)

---

The [Smart.Braces](#) tool can be customized via the Editor Options dialog box.

As soon as Productivity! is installed, additional node appears in the *Editor Options* tree view (*Editor Options* / *Editor* property page)

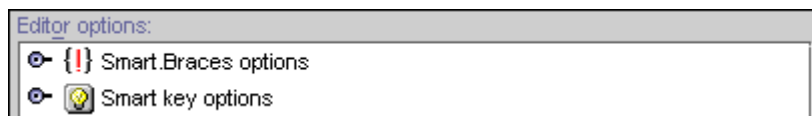


Figure 100 Smart.Braces Options

This node contains the [Smart.Braces](#) customization options. You may fully customize all the features of [Smart.Braces](#) to satisfy your needs and goals.

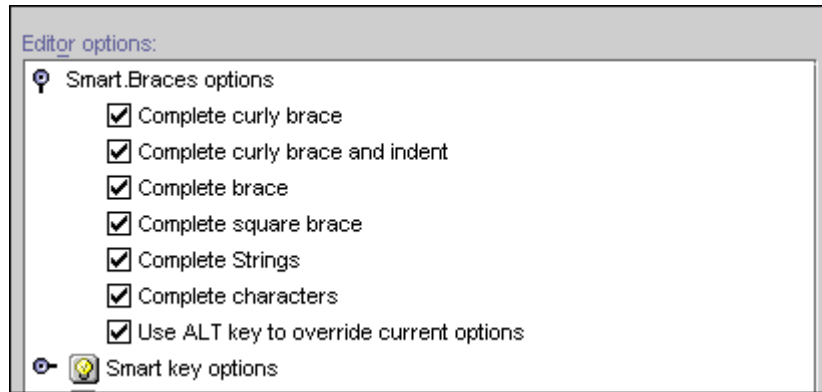


Figure 101 Smart.Braces Options (Expanded)

The following options are available:

*Complete curly brace*

Allows you to specify whether [Smart.Braces](#) should complete the curly brace ( { } )

*Complete curly brace and indent*

Allows you to specify whether [Smart.Braces](#) should complete the curly brace ( { } ) and make the indent in accordance with the currently set size

*Complete brace*

Allows you to specify whether [Smart.Braces](#) should complete the brace ( ( ) )

*Complete square brace*

Allows you to specify whether [Smart.Braces](#) should complete the brace ( [ ] )

*Complete Strings*

Allows you to specify whether [Smart.Braces](#) should complete the string constants ( " )

*Complete characters*

Allows you to specify whether [Smart.Braces](#) should complete the character constants ( ' )

*Use ALT key to override the current options*

With this option enabled, the current options can be overridden provided the ALT key is pressed when typing. For example, if Complete curly brace option is enabled, pressing { with the **ALT** key will only insert the opening curly brace without the corresponding closing one.

**NOTE:**

If a non-standard JBuilder keymap is used (such as Vi/VIM), the [Smart.Braces](#) may conflict with the keymap settings. Apparently, for keymap VI the ' and " symbols may be overridden by Smart.Braces. It is justified by the features of the vi implementation (improper implementation of the Keymap default action).

However the part of [Smart.Braces](#) functionality that causes the conflict can be disabled. To do this, you should add the following lines into your JBuilder.config file (placed in JBuilder/bin directory):

```
vmparam -DProductivity.Smart.Braces.CompleteCharacters=no
```

```
vmparam -DProductivity.Smart.Braces.CompleteStrings=no
```

**General Page**

---

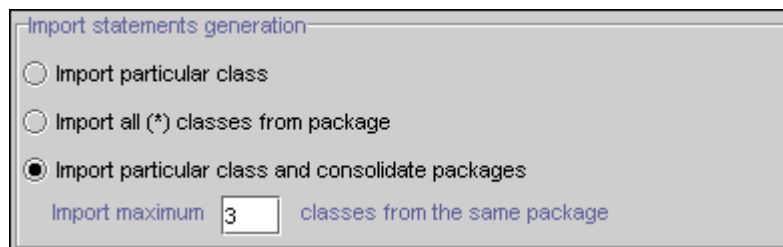
The *General* page of the *Productivity! Editor Options* page provides the following options:

- *Import statement generation*
- *Search options*
- *Sort classes by*
- *Autocomplete*

**Import statements generation**

Productivity! contains a number of tools designed for imports modification - [Class.Insight](#), [Implement.Insight](#), [Override.Insight](#), [Constructor.Insight](#), [Imports.Beautify](#) and [Smart.Instantiate](#).

All these tools share common settings for the import statements modification so modification of these options will affect all tools mentioned above.



**Figure 102 Import statements generation options**

The following options are available for management of import statement modifications:

*Import particular class*

If this option is turned on, the import statement for the required class will be inserted, however imports consolidation will not be applied.

*Import all (\*) classes from package*

If this option is turned on, all (\*) classes from all the packages will be imported and particular imports from the same package will be removed. This option is useful when a large amount of classes from the same package are used.

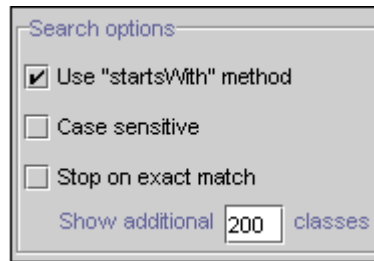
*Import particular class and consolidate packages*

If this option is turned on, a particular class will be imported if the number of imports from the same package does not exceed the maximum allowed. The maximum amount of classes to be imported without import statements consolidation is controlled by the "Import maximum N classes from the same package field."

If the number of imports from the same packages exceeds the specified limit, all the imports of a particular class from the required package will be removed and import statement for the whole (\*) package will be inserted instead.

**Search options**

These options allow you to tune the algorithm used for search of items within the Productivity! popup lists.



**Figure 103 Search options**

The following options are available for search control:

*Use "startsWith" method*

If this option is turned on, all the search operations will be performed for the string with the value of the word at cursor. Otherwise, the search will include the strings that contain the required substring (usually a word at cursor).

*Case sensitive*

If this option is turned on, the case sensitive search algorithm will be used.

*Stop on exact match*

If this option is turned on, only classes with the names that exactly match the word at cursor will be shown.

*Show additional classes*

If the *Stop on exact match* option is disabled, you may specify the amount of classes you would like to see in the popup list. Since the overall amount of classes may be quite important, you may make the list of classes less extended.

**NOTE:** This option is common for the following Insights: [Class.Insight](#), [Browse.Insight](#), [Implement.Insight](#) and [Smart.Instantiate](#)

---

**Sort classes by**

These options allow controlling the sorting of classes for the following Insights: [Class.Insight](#), [Browse.Insight](#), [Implement.Insight](#) and [Smart.Instantiate](#).

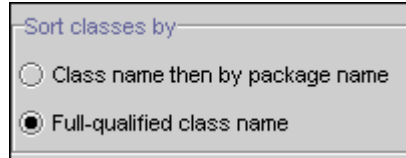


Figure 104 Sort Classes By options

The following options are available for classes sorting control:

*Class name then by package name*

If you select this option, the classes will be sorted according to the class name and then class package;

*Full-qualified class name*

If you select this option, the classes will be sorted according to their full-qualified names.

---

**Autocomplete**

Productivity! allows automatic execution of the Insight primary action when there is only one possible variant found. In this case the action will be performed without the Insight popup window being shown.

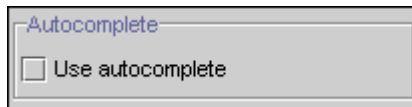


Figure 105 Autocomplete options

*Use autocomplete*

If this checkbox is enabled, the autocomplete option will be turned on, and Productivity! will automatically complete the actions if possible.

## Usage Page

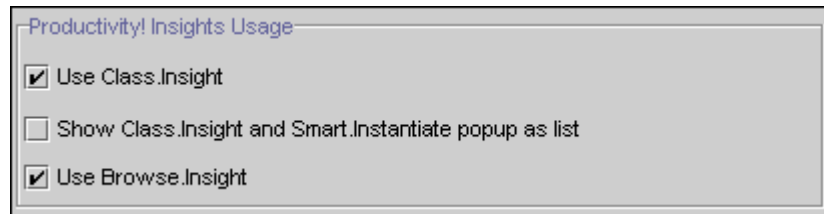
---

The General page of the Productivity! Editor Options page provides the following options:

1. *Productivity! Insights usage*
2. *Invoke insights during debugging*
3. *Help.Insight*
4. *Highlight.Navigate popup hiding mode*
5. *Superclass Changing Policy*

### Productivity! Insights Usage

Two tools included into Productivity! - [Class.Insight](#) and [Browse.Insight](#) - use the same default shortcuts (**Ctrl+Alt+Space** and **Ctrl+Minus**, respectively (CUA)) as JBuilder built-in tools. If you want to continue using the JBuilder built-in tools you may disable the Productivity! insights startup using this option.



**Figure 106 Productivity! Insights Usage options**

#### *Use Class.Insight*

If this checkbox is not selected, the original JBuilder tools will be invoked instead of Productivity! [Class.Insight](#) by pressing the appropriate shortcut.

#### *Show Class.Insight and Smart.Instantiate popup as list*

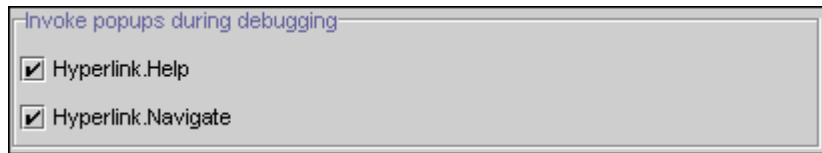
If this checkbox is selected, the popup window used by [Class.Insight](#) and [Smart.Instantiate](#) will not include the Navigation Pane and will be similar to JBuilder built-in Member Insight.

#### *Use Browse.Insight*

If this checkbox is not selected, the original JBuilder tool will be invoked instead of Productivity! [Browse.Insight](#) by pressing the appropriate shortcut.

**Invoke popups during debugging**

During debugging, JBuilder provides ability to inspect the symbol under cursor using the appropriate popup window. Since both JBuilder and Productivity! [Hyperlink.Navigate](#) windows are invoked by placing mouse over the symbol with the **CTRL** key pressed down, in some cases these windows may overlap.

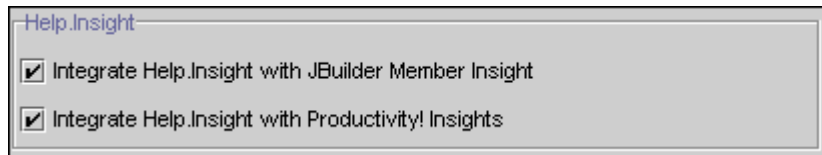


**Figure 107 Invoke popups during debugging options**

To eliminate this, you may disable [Hyperlink.Help](#) and [Hyperlink.Navigate](#) popup windows if there is an active debugging session.

**Help.Insight**

You may specify whether [Help.Insight](#) should be integrated with another insights.



**Figure 108 Help.Insight options**

*Integrate Help.Insight with JBuilder Member Insight*

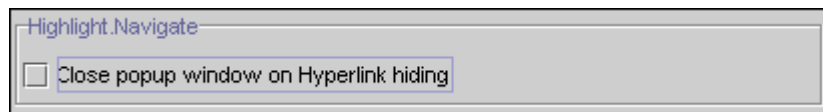
If this checkbox is selected, [Help.Insight](#) will be integrated with JBuilder Member Insight. In such mode, as soon as you change the selection within the Member Insight popup window list, [Help.Insight](#) with the help for a selected item will appear near the Member Insight popup window.

*Integrate Help.Insight with Productivity! Insights*

If this checkbox is selected, [Help.Insight](#) is integrated with all the insights included into Productivity!. Please note that even if you disable this integration, you will still be able to invoke [Help.Insight](#) for a selected item in the insight popup list. To do this, you just need to press the shortcut key normally used for [Help.Insight](#) invocation (the default shortcut is **CTRL+F1** under CUA) when Productivity! insight is being used.

**Hyperlink popup hiding mode**

This option allows you to hide the [Hyperlink.Navigate](#) popup window.



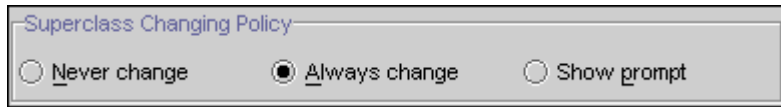
**Figure 109 Highlight.Navigate options**

If selected, the popup window will be hidden together with the hyperlink. If not, the popup window will be closed in accordance with the delay specified on the Delays page.



**Super Class Changing Policy**

This option allows you to specify how the Productivity! [Implement.Insight](#) tool will handle the situation when an extra super class is assigned to a class.



**Figure 110 Superclass Changing Policy options**

In such case [Implement.Insight](#) will perform the following actions depending on the currently selected value:

*Never change* - [Implement.Insight](#) will not change the super class and will not implement methods from the proposed super class.

*Always change* - [Implement.Insight](#) will change the super class to the selected one and will override all abstract methods.

*Show prompt* - [Implement.Insight](#) will show a prompt dialog allowing you to specify what should be done.

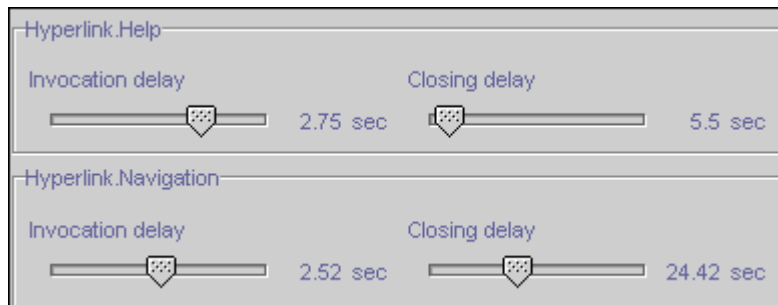
**Delays Page**

The Delays page of the Productivity! Editor Options page provides the following options:

1. *Hyperlink.Help Delays*
2. *Hyperlink.Navigate Delays*
3. *Help.Insight Delay*
4. *Context Discovering Timeout*

**Hyperlinks Delays**

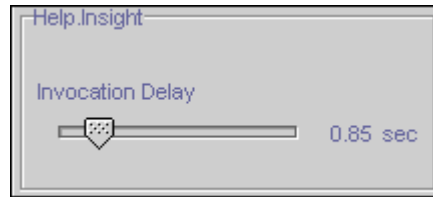
With these options you may specify the delays used for invocation and closing of popup windows displayed by the [Hyperlink.Help](#) and [Hyperlink.Navigate](#) tools.



**Figure 111 Hyperlinks options**

**Help.Insight  
Delay**

With this option you may specify the [Help.Insight](#) delay for JBuilder built-in Member Insight as well as for Productivity! insights.



**Figure 112 Help.Insight options**

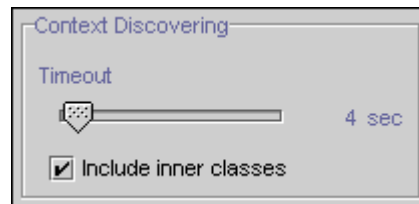
You may also indicate whether [Help.Insight](#) will be integrated with the JBuilder built-in insights using the *Enable Help for built-in JBuilder insights* checkbox. From the *Enable Help for Productivity! Insights* option you can specify whether or not [Help.Insight](#) will be integrated with the insights included into Productivity!.

*Invocation delay*

With this slider you may define the timeout between the time when a member in Member Insight (or Productivity! Insights) is selected and when the [Help.Insight](#) popup window is displayed.

**Context  
Discovering**

In certain cases, (particularly, for the classes with a large number of inner classes), [Context.Insight](#) may display only the upper class information.



**Figure 113 Context Discovering options**

The reason for this is the limitation of JBuilder JOT subsystem that requires significant amount of time (up to tens of seconds) to retrieve information about the inner classes. To avoid hang-up of JBuilder, the time required for collection of context information was limited to 2 seconds. Thus, if JOT fails to provide the data within this interval, only the upper class information is selected. Relatively slow performance of [Context.Insight](#) when cursor is placed on the white space between class methods can also be justified by these reasons. The same limitations may affect other tools that use the same functionality (Override.Insight, Implement.Insight).

From this option you can specify the maximum time required to discover the context.

**Tools Page – Pro!**

The Tools page of the Productivity! Editor Options page provides the following options:

1. *Smart.Clipboard;*

2. *Smart.Braces*;

**Smart.Clipboard** The *Smart.Clipboard* group allows performing tuning of Smart.Clipboard Tools.

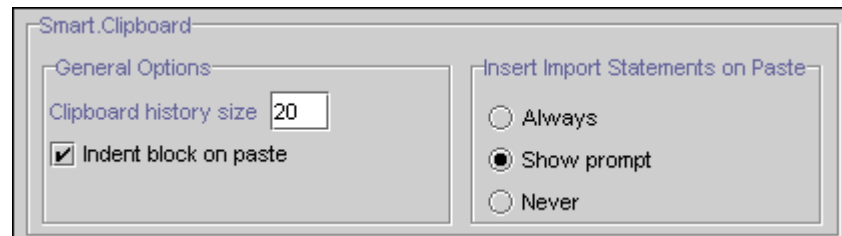


Figure 114 Smart.Clipboard options

*General Options*

This options group provides ability to specify size of clipboard history (clipboard buffer) and enable/disable automatic indentation of the inserted block according to current indent level (this option is applicable to pasting Java code into Java files).

*Insert Import Statements On Paste*

This set of radio buttons provides ability to specify policy for import statements generation that should be used during pasting of Java code block (this option is applicable to pasting Java code into Java files). If this option is not disabled, [Smart.Clipboard](#) will optionally insert appropriate import statements for all classes found in the copied code fragment.

**Smart.Braces** The *Smart.Braces* options group provides ability to customize behavior of [Smart.Braces](#) and [Matching.Code.Highlight](#) tools.

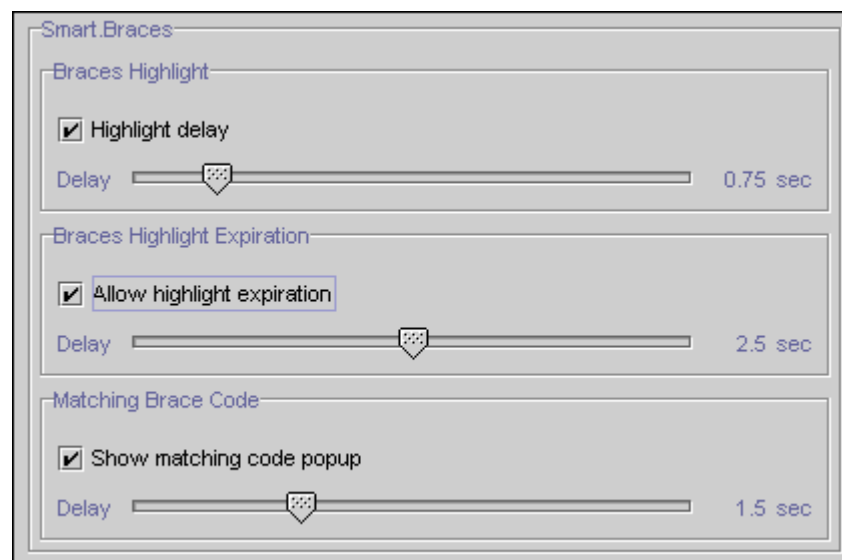


Figure 115 Smart.Braces options

Using this options group, the user is able to specify delays used for highlight, expiration and matching code popup displaying.

**Assistant Page – Pro!**

---

The Assistant page of the Productivity! Editor Options page is used for customizing of Productivity! Assistant and provides the following options:

1. Issue Highlight Style
2. Assistant Popup Delay
3. Auto Fix Errors
4. Rename Assistant

**Issue Highlight Style**

The *Issue Highlight Style* options group provides ability to specify line style and color for visual marks those are used to highlight issue in editor.

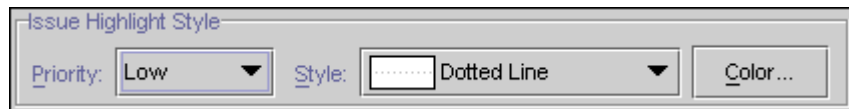


Figure 116 Assistant Issue Highlight Style options

**Assistants Displaying Delay**

The *Assistants Displaying Delay* option allows specifying delay should be used to display Assistant popup.

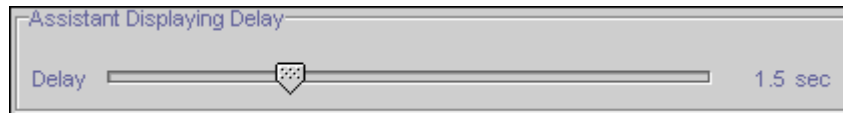


Figure 117 Assistant Displaying Delay

**Auto Fix**

The Auto Fix Errors options allow enabling/disabling auto fix functionality for all or particular types of errors.

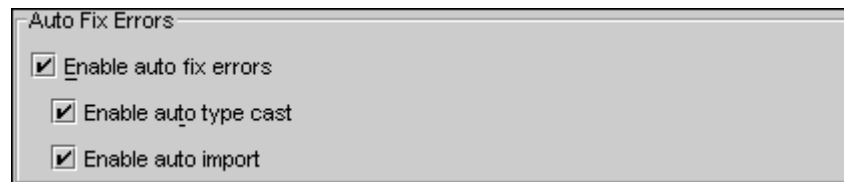


Figure 118 Auto Fix Options

**Rename Assistant**

The Rename Assistant options allows enabling/disabling rename functionality for all or particular types of entities. It's also possible to specify shortcut to start refactoring using these options. If the Enter key is chosen to start refactoring the Ctrl+Enter key allows simple renaming and vice versa.

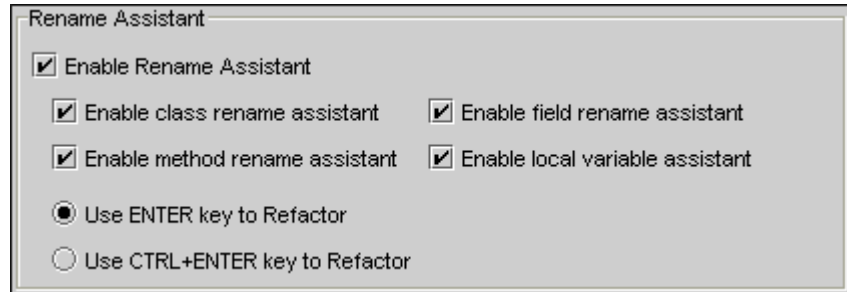


Figure 119 Rename Assistant Options

**Smart.JavaDoc Page – Pro!**

The Smart.JavaDoc page of the Productivity! Editor Options page is used for customizing of the Smart.JavaDoc tool and provides the following options:

1. *Issue Highlight Style*
2. *Java Doc Generation Style*

**Issue Highlight Style**

The *Issue Highlight Style* options group provides ability to customize style and color of marks should be used to visually highlight errors in JavaDoc comments (such as missing or redundant parameters, missing throws, unknown tags etc).

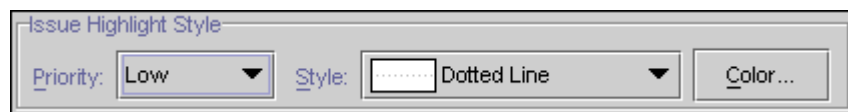


Figure 120 Smart.JavaDoc Issue Highlight options

**JavaDoc Generation Style**

The JavaDoc Generation style options group provides ability to customize style of JavaDoc comment generated by Smart.JavaDoc.

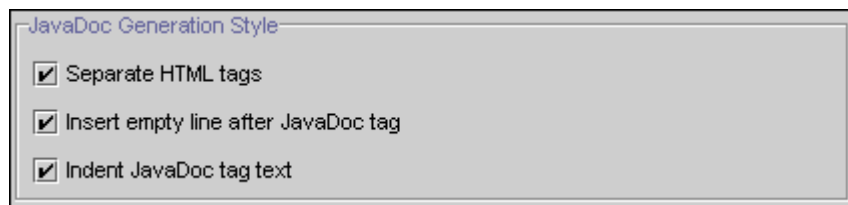


Figure 121 JavaDoc Generation Style options

There are the following options:

- *Separate HTML tags* - if enabled, all tags such as `<p>` will be placed on different lines;
- *Insert empty line after JavaDoc tag* - if enabled, the empty line will be inserted between JavaDoc tags;
- *Indent JavaDoc tag text* – if enabled, the multi-line tag description will be indented from left by the tag itself;

---

### Smart.Templates Page – *Pro!*

This page allows customizing of [Smart.Templates](#) behavior and maintaining the templates list.

It provides the following options:

1. *Scope*
2. *Template*
3. *Code*
4. *Quick Expand Key*

---

#### Scope

The *Scope* combobox allows selection of template list to maintain.

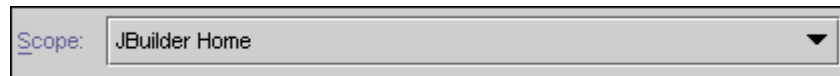


Figure 122 Templates List Scope options

The following scopes are currently supported.

- *User Home* – specifies the template list that belongs to the currently logged in user. All the templates belonging to this scope are stored in the files `<Group>.templates` located in the `<User Home>/jbuilder<N>` folder.
- *JBuilder Home* – specifies the template list that is JBuilder wide so it is available for all users those use this installation of JBuilder. All the templates belonging to this scope are stored in the files `<Group>.templates` located in the `<JBuilder Install Folder>/Productivity` folder.

All the templates from all available scopes are merged allowing transparent usage of templates defined in any scope. Templates defined in the *User Home* scope have priority over ones defined in *JBuilder Home* one.

---

#### Templates

The *Templates* table shows the list of templates available in currently selected scope

and it allows selection of particular template to maintain.

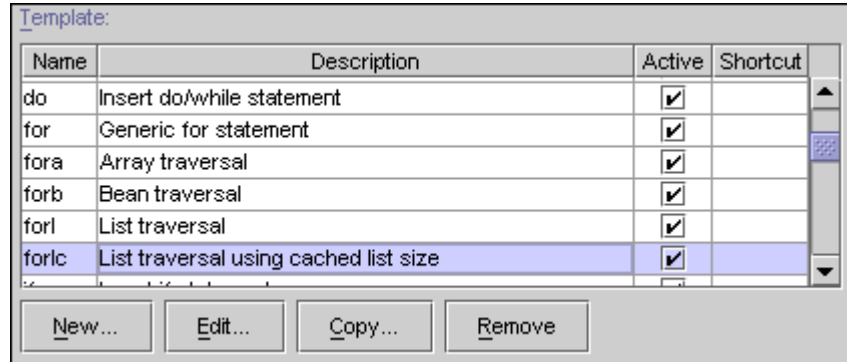


Figure 123 Templates list

#### *New*

The *New* button allows creation of a new template. After pressing of this button the *Edit Template Dialog* is showing which allows specifying miscellaneous attributes of newly created template.

#### *Edit*

The *Edit* button allows maintenance of currently selected template. After pressing of this button the *Edit Template Dialog* is showing which allows specifying miscellaneous attributes of selected template.

#### *Copy*

The *Copy* button allows creation of a new template using currently selected template as a prototype. After pressing of this button the *Edit Template Dialog* is showing which allows specifying miscellaneous attributes of newly created template.

#### *Remove*

The *Remove* button allows removal of currently selected template.

---

#### **Code**

The *Code* editor pane provides preview of currently selected template.

```
Code:
int size = #list#.size();
for (int #index# = 0; #index# < size; #index#++) {
    #varType# #var# = (#varType#) #list#.get(#index#);
    #|##selBlock#
}
```

Figure 124 Template Code preview

---

**Quick Expand Key** This options group allows selection of key can be used to quickly expand a template which name is match to the word at caret.

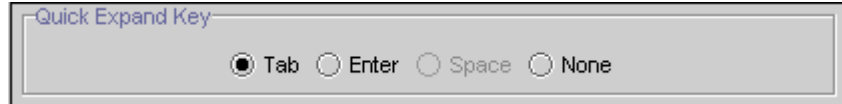


Figure 125 Quick Expand Key options

## Edit Template Dialog – *Pro!*

---

The *Edit Template Dialog* allows maintenance of a template and specifying its structure and properties.

It contains the following pages:

- *General* – one that allows maintaining general properties of template;
- *Options* – one that allows specifying miscellaneous properties of template;
- *Fields* – one that allows maintaining the fields belonging to template;
- *Shortcuts* – one allows specifying of shortcuts for each supported keymap.

### General Page

---

The *General* page allows maintenance of template general properties.



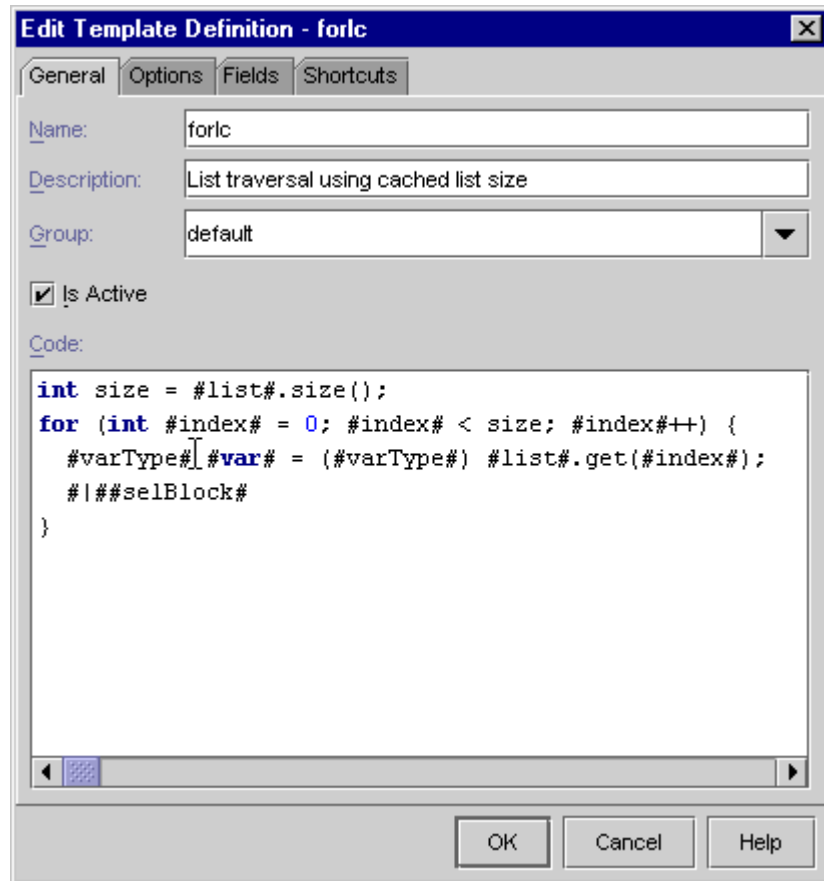


Figure 126 Edit Template Definition Dialog – General Page

It provides the following options:

#### *Name*

The *Name* edit box allows specifying the name for the template. This name would be short as possible though it should be meaningful enough to recognize the template. Please note that template name should be unique within current scope.

#### *Description*

The *Description* edit box allows specifying the description for the template.

#### *Group*

The *Group* combobox allows specifying the group this template belongs to. The group defines the name of the file in which this template will be stored after saving of the template list. Allocation of templates across scopes and groups allows better controlling them and provides easy way to share them between team members.

#### *Active*

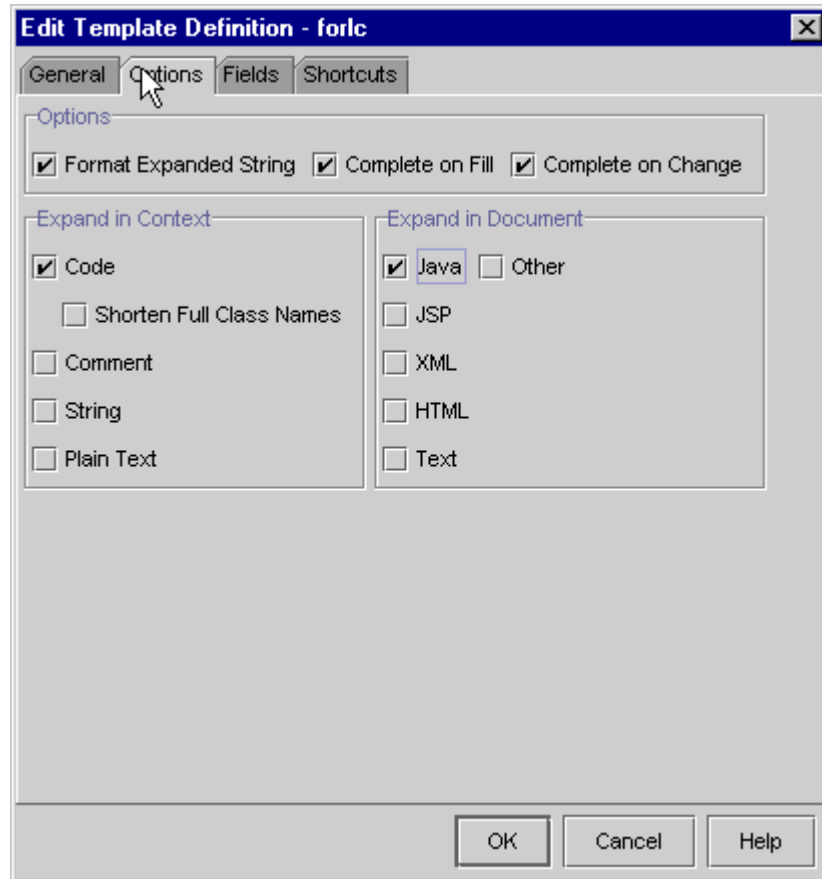
The *Active* checkbox allows specifying whether this template is available for use.

*Code*

The *Code* editor pane allows typing the code of the template. The code of the template consists of code fragments and optionally template fields within them. Each field is represented by its name enclosed by # sign. In general, it's enough just to state fields in the template code and they will be automatically added to the template. To specify properties for the desired template fields the *Fields* page is used.

**Options Page**

The *Options* page allows specifying miscellaneous properties of template.



**Figure 127 Edit Template Definition Dialog – Options Page**

**Options**

This group provides options those control general template behavior.

*Format Expanded String*

The *Format Expanded String* option specifies whether template code should be formatted and indented according to the current indent level and code style.

*Complete on Fill*

The *Complete on Fill* option allows specifying whether the running template should be completed when all template fields are filled.

*Complete on Change*

The *Complete on Change* option allows whether the running template should be completed when there is any change occurred in the document outside of template.

---

**Expand in Context**

This group provides options those allow specifying the context applicable to expand this template.

*Code*

This option specifies whether this template can be expanded in Java code (reserved words, identifiers, expressions etc.).

*Shorten Full Class Names*

This option specifies whether all known classes' references from the template code should be shorten as well as appropriate import statements should be added.

---

**Expand in Document**

This group provides options those allow specifying the document type applicable to expand this template.

---

**Fields Page**

The *Fields* page allows maintaining the fields are belonging to the template is being editing.

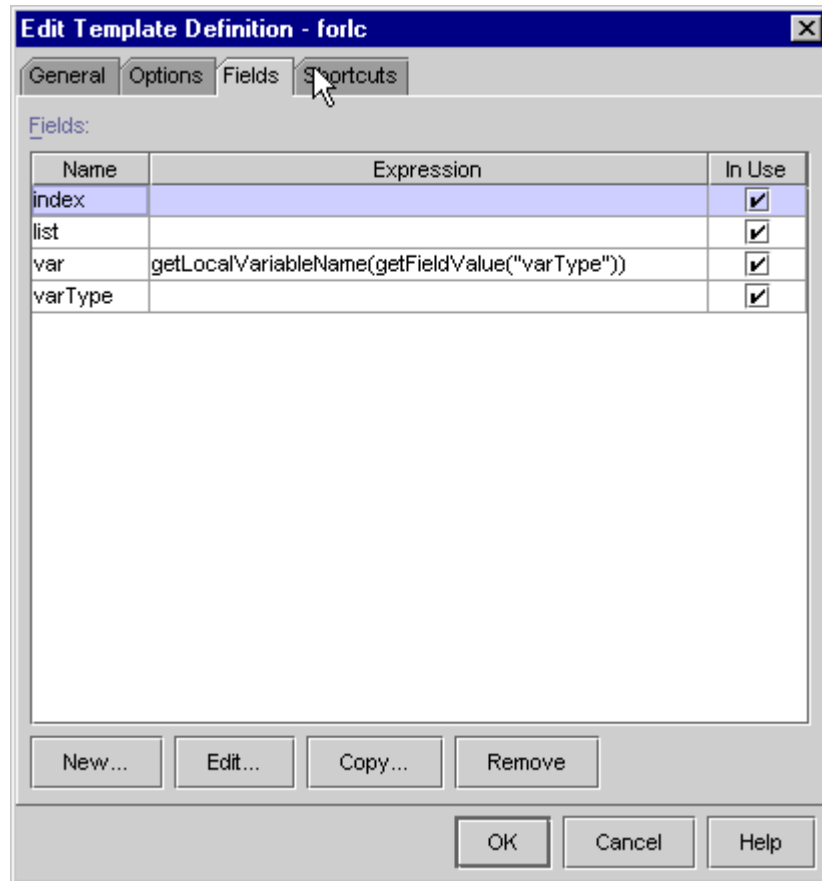


Figure 128 Edit Template Definition Dialog – Fields Page

### *Fields*

The *Fields* table shows the list of fields defined in the template and it allows selection of particular field to maintain. In this table each field occupies one row and each table column represents particular fields' attribute. The *Name* and *Expression* columns show the name and calculate expression for the field. The *In Use* column shows whether this field is used somewhere in the template code.

### *New*

The *New* button allows creation of a new field.

### *Edit*

The *Edit* button allows maintenance of currently selected field.

### *Copy*

The *Copy* button allows creation of a new field using currently selected field as a prototype.

*Remove*

The *Remove* button allows removal of currently selected field.

**Shortcuts Page**

The *Shortcuts* page allows assigning keyboard shortcuts to the template for the each key maps installed in the JBuilder.

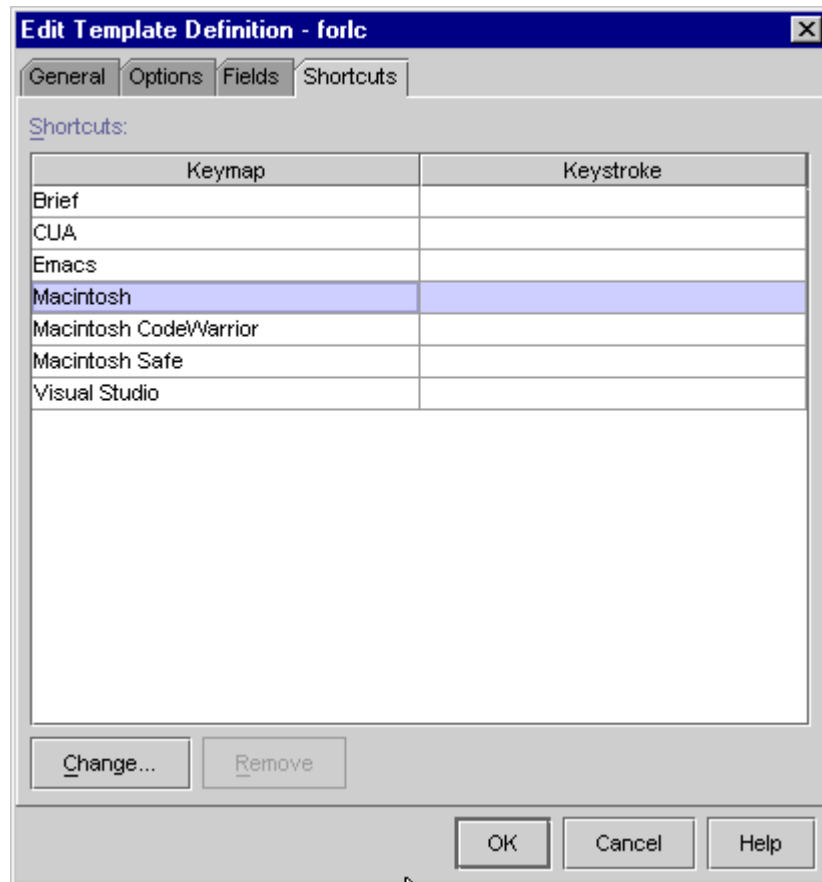


Figure 129 Edit Template Definition Dialog – Shortcuts Page

*Shortcuts*

The *Shortcuts* table shows the list of key maps along with shortcuts assigned to the template for the each particular key map.

*Change*

The *Change* button allows assigning or changing the shortcut for the selected key map. On pressing this button the *Assign Key Stroke* dialog is showing that allows defining the shortcuts attributes.

*Remove*

The *Remove* button allows removal of shortcut for the selected key map.

## Edit Template Field Dialog – *Pro!*

This dialog allows maintenance of template field properties.

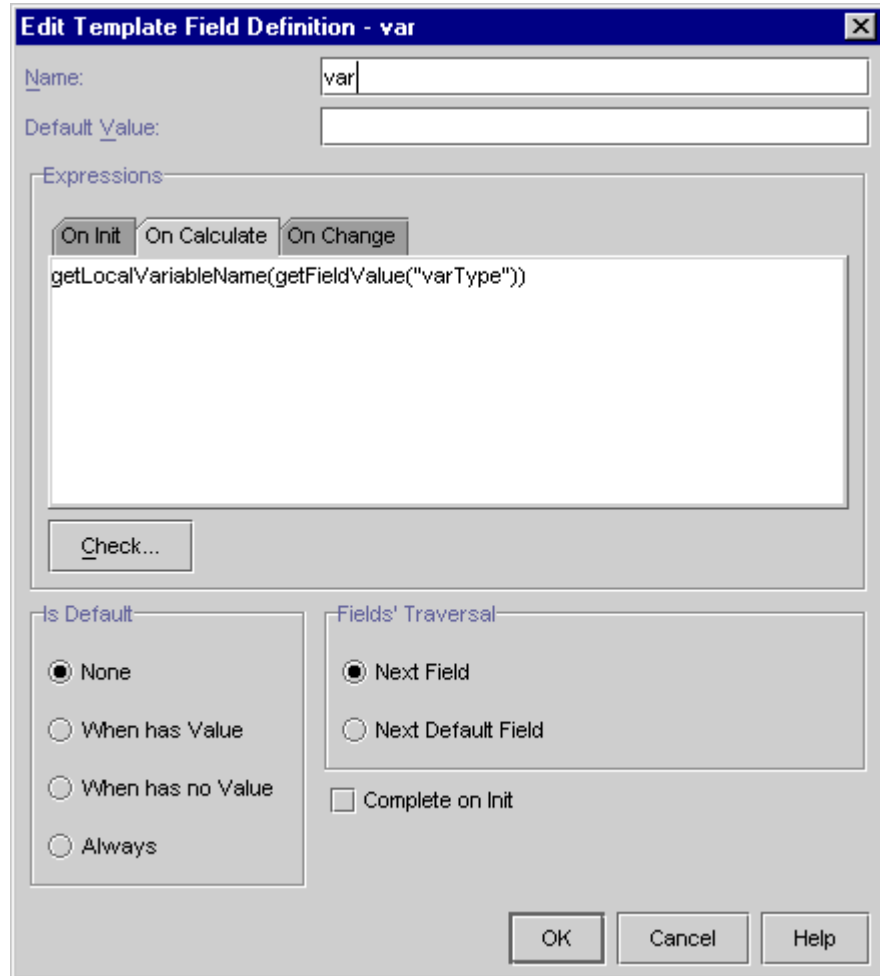


Figure 130 Edit Template Field Definition Dialog

The following options are available:

### General

#### *Name*

The *Name* field allows specifying the name of the template field. Please note that this name should be unique within particular template.

#### *Default Value*

The *Default Value* field allows specifying the initial value should be assigned to the field on template execution.

*Complete on Init*

The *Complete on Init* checkbox specifies whether the field should be completed if it gets a value after initialization. Completed fields are represented in the running template as simple fragments of code rather than editors.

---

**Expressions**

This group allows specifying the expressions applicable to different stages of template field lifecycle. The *Check* button allows checking entered expressions to avoid syntax errors.

*On Init*

The *On Init* editor pane allows specifying the expression will be evaluated to assign initial value for the template field.

*On Calculate*

The *On Calculate* editor pane allows specifying the expression will be evaluated to assign value for the template field to reflect changes in this one or in other fields.

*On Change*

The *On Change* editor pane allows specifying the expression will be executed to make some actions to reflect value changes of this field. Please note that if this expression returns some value, this value will be ignored so execution of this expression can't change any fields anyway.

---

**Is Default**

This group allows defining the policy and conditions used to determine whether this field should be first focused one on template expand. The following options are available:

*None*

If this option is turned on, this field can be the first focused field only if there are no other candidate fields.

*When has Value*

If this option is turned on, this field can be the first focused field only if it has value after initialization.

*When has no Value*

If this option is turned on, this field can be the first focused field only if it has no value after initialization.

*Always*

If this option is turned on, this field can be the first focused field without any conditions.

**Fields'  
Traversal**

---

This group defines which field should be focused when the user presses the **Enter** key within this field. The following options are available:

*Next Field*

If this option is turned on, the next field with the different name will be focused.

*Next Default Field*

If this option is turned on, the field to be focused will be determined using the Default Policy of the rest of fields.



## IDE Options Dialog

The *IDE Options* dialog includes a property page added by Productivity!, which allows user to select the Metal theme to be used by JBuilder and customize Task List Reminders options.

### Look&Fill Improvements

The *Look&Fill Improvements* group provides the themes to select from for the Metal look and feel.

Productivity! allows you to customize the current theme of Swing Metal LF. There are two additional themes added - Plain Steel and Plain Steel (W2K).

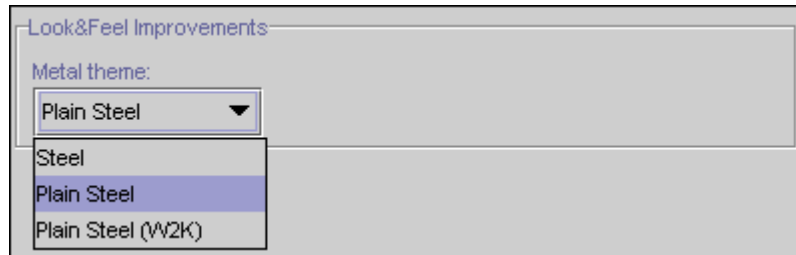


Figure 131 Metal theme options

Below you can see the samples of UI under different themes.

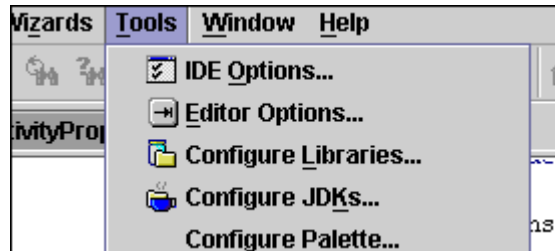


Figure 132 Default Steel Metal Theme sample



Figure 133 Plain Steel Theme sample

If you are using Metal LF, you may select one of the themes provided. Plain themes are similar to the standard one as they are derived from it, however the bold attributes of fonts were removed and the font size was slightly decreased. Plain Steel (W2K), in

addition, sets fonts to the mimic ones used in Windows 2000 (Tahoma) and therefore requires this font to be installed.

**Task List Reminders - Pro!**

The *Task List Reminders* groups allows you to tune how Productivity! should handle reminders defined in the Task List.

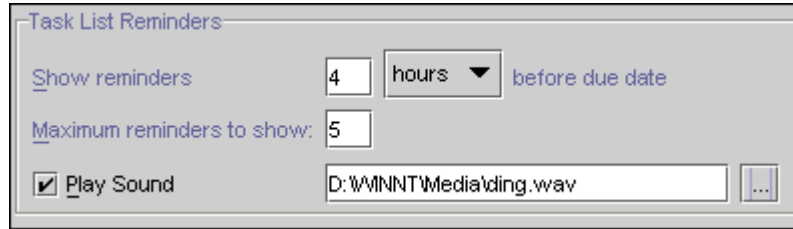


Figure 134 Task List Reminders Group

This group includes the following options:

*Show reminders [] before due date*

Allows to specify time interval should be used to show reminder before reminder's task due date. As units of time interval measurement minutes, hours or days may be selected.

*Maximum reminders to show*

Allows specifying the maximal amount of reminders those may be simultaneously visible.

*Play Sound*

Provides ability to specify that some sound should be played on reminder displaying and assign appropriate sound file should be used by reminder.

## Productivity! Key Bindings

---

Productivity! supports all JBuilder built-in keymaps, such as:

1. Brief
2. CUA
3. Emacs
4. Macintosh (Mac)
5. Macintosh Code Warrior (Mac CW)
6. Visual Studio (VS)

Also, an additional keymap is provided – Mac Safe, intended to improve Productivity! usability on the Macintosh platform.

**NOTE:** If you use Professional or Enterprise edition of JBuilder, you are able to customize these shortcuts by using either *Editor Options | Editor | Keymap | Customize...* or *IDE Options | Browser | Keymap | Customize...* dialogs. All the Productivity! shortcuts are placed in the Productivity! group.

## Key Bindings for CUA, Brief and Visual Studio keymaps

The following table outlines the shortcuts to Productivity! features. For a detailed description of these features please see Productivity! Tools.

**Table 6 Productivity! Key Bindings for CUA, Brief and Visual Studio keymaps**

| Tool                   | CUA            | Brief            | VS             |
|------------------------|----------------|------------------|----------------|
| Class.Insight          | Ctrl+Alt+Space | Ctrl+Alt+Space   | Ctrl+Alt+Space |
| Class.Insight          | Ctrl+Alt+H     | Ctrl+Alt+H       | Ctrl+Alt+H     |
| Browse.Insight         | Ctrl+Minus     | Ctrl+Shift+Minus | Ctrl+Minus     |
| Browse.Members         | Alt+Minus      | Ctrl+Alt+Minus   | Alt+Minus      |
| Help.Insight           | Shift+F1       | Shift+F1         | Shift+F1       |
| Help.Insight.OnMembers | Alt+F1         | Alt+F1           | Alt+F1         |
| Implement.Insight      | Ctrl+Alt+I     | Ctrl+Alt+I       | Ctrl+Alt+I     |
| Override.Insight       | Ctrl+M         | Ctrl+M           | Ctrl+M         |
| Constructor.Insight    | Ctrl+Shift+M   | Ctrl+Shift+M     | Ctrl+Shift+M   |
| Context.Insight        | Ctrl+Q         | Ctrl+Q           | Ctrl+Q         |
| Imports.Beautify       | Ctrl+Alt+B     | Ctrl+Alt+B       | Ctrl+Alt+B     |
| Smart.Instantiate      | Alt+I          | Ctrl+Shift+I     | Alt+I          |
| Hyperlink.Navigate     | Ctrl+MOUSE     | Ctrl+MOUSE       | Ctrl+MOUSE     |
| Hyperlink.Help         | Alt+MOUSE      | Alt+MOUSE        | Alt+MOUSE      |
| Easy.JavaDoc           | Ctrl+D         | Ctrl+Alt+D       | Ctrl+Alt+D     |
| Easy.JavaDoc.Insight   | Ctrl+Shift+D   | Ctrl+Shift+D     | Ctrl+Shift+D   |
| GetSet.Creator         | Alt+Shift+A    | Ctrl+Shift+A     | Alt+Shift+A    |
| Get.Creator            | Alt+Shift+G    | Ctrl+Shift+G     | Ctrl+Shift+G   |
| Set.Creator            | Alt+Shift+S    | Ctrl+Shift+S     | Ctrl+Shift+S   |

Table 7 Productivity! Pro Key Bindings for CUA, Brief and Visual Studio keymaps

| Tool                           | CUA                 | Brief               | VS                  |
|--------------------------------|---------------------|---------------------|---------------------|
| Assistants                     | Alt+Enter           | Alt+Enter           | Alt+Enter           |
| Delegate.Insight               | Alt+Shift+M         | Alt+Shift+M         | Alt+Shift+M         |
| Find.Matching.Brace            | Ctrl+\              | Ctrl+\              | Ctrl+\              |
| Find.Matching.Code             | Ctrl+Shift+\        | Ctrl+Shift+\        | Ctrl+Shift+\        |
| Introduce.Constructors         | Alt+Shift+C         | Alt+Shift+C         | Alt+Shift+C         |
| Navigator.Insight              | Alt+Shift+N         | Alt+Shift+N         | Alt+Shift+N         |
| Navigator.Next                 | Ctrl+Page Down      | Alt+Page Down       | Ctrl+Page Down      |
| Navigator.Previous             | Ctrl+Page Up        | Alt+Page Up         | Ctrl+Page Up        |
| Persistent.Bookmarks.Navigate  | Alt+Shift+B         | Alt+Shift+B         | Alt+Shift+B         |
| Select.Class                   | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Select.CodeBlock               | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Select.Method                  | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Select.Statement               | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Selection.Narrow               | Ctrl+Shift+W        | Ctrl+Shift+W        | Ctrl+Shift+W        |
| Selection.Expand               | Ctrl+W              | Alt+Shift+W         | Ctrl+W              |
| Smart.Clipboard.Insight        | ALT+Shift+V         | ALT+Shift+V         | ALT+Shift+V         |
| Smart.Clipboard.PopPaste       | Ctrl+Alt+Insert     | Ctrl+Alt+Insert     | Ctrl+Alt+Insert     |
| Smart.Clipboard.Swap           | Ctrl+Shift+Insert   | Ctrl+Shift+Insert   | Ctrl+Shift+Insert   |
| Smart.Templates.Expand         | Ctrl+J              | Ctrl+J              | Ctrl+J              |
| Smart.Templates.ExpandLast     | Ctrl+Shift+J        | Ctrl+Shift+J        | Ctrl+Shift+J        |
| Smart.Templates.ExpandOnTheFly | Alt+Shift+J         | Alt+Shift+J         | Alt+Shift+J         |
| Highlight Local References     | Ctrl+Alt+Enter      | Ctrl+Alt+Enter      | Ctrl+Alt+Enter      |

## Key Bindings for Emacs, Macintosh and Macintosh Code Warrior keymaps

The following table outlines the shortcuts to Productivity! features. For a detailed description of these features please see Productivity! Tools.

**Table 8 Productivity! Key Key Bindings for Emacs, Macintosh and Macintosh Code Warrior keymaps**

| Tool                   | Emacs          | Mac            | Mac CW         | Mac Safe       |
|------------------------|----------------|----------------|----------------|----------------|
| Class.Insight          | Ctrl+Alt+Space | Ctrl+Alt+Space | Ctrl+Alt+Space | Ctrl+Alt+Space |
| Class.Insight          | Ctrl+Alt+H     | Ctrl+Alt+H     | Ctrl+Alt+H     | Ctrl+Alt+H     |
| Browse.Insight         | Ctrl+Minus     | Ctrl+Minus     | Ctrl+Minus     | Ctrl+Minus     |
| Browse.Members         | Alt+Minus      | Alt+Minus      | Alt+Minus      | Alt+Minus      |
| Help.Insight           | Shift+F1       | Shift+F1       | Shift+F1       | Shift+F1       |
| Help.Insight.OnMembers | Alt+F1         | Alt+F1         | Alt+F1         | Alt+F1         |
| Implement.Insight      | Ctrl+Alt+I     | Ctrl+Alt+I     | Ctrl+Alt+I     | Ctrl+Alt+I     |
| Override.Insight       | Ctrl+Alt+M     | Ctrl+M         | Ctrl+M         | Ctrl+M         |
| Constructor.Insight    | Ctrl+Shift+M   | Ctrl+Shift+M   | Ctrl+Shift+M   | Ctrl+Shift+M   |
| Context.Insight        | Ctrl+Q         | Ctrl+Q         | Ctrl+Q         | Ctrl+Q         |
| Imports.Beautify       | Ctrl+Shift+B   | Ctrl+Alt+B     | Ctrl+Alt+B     | Ctrl+Alt+B     |
| Smart.Instantiate      | Alt+I          | Alt+I          | Alt+I          | Alt+I          |
| Hyperlink.Navigate     | Ctrl+MOUSE     | Ctrl+MOUSE     | Ctrl+MOUSE     | Ctrl+MOUSE     |
| Hyperlink.Help         | Alt+MOUSE      | Alt+MOUSE      | Alt+MOUSE      | Alt+MOUSE      |
| Easy.JavaDoc           | Ctrl+Alt+D     | Ctrl+D         | Ctrl+D         | Ctrl+D         |
| Easy.JavaDoc.Insight   | Ctrl+Shift+D   | Ctrl+Shift+D   | Ctrl+Shift+D   | Ctrl+Shift+D   |
| GetSet.Creator         | Alt+Shift+A    | Alt+Shift+A    | Alt+Shift+A    | Alt+Shift+A    |
| Get.Creator            | Alt+Shift+G    | Alt+Shift+G    | Alt+Shift+G    | Alt+Shift+G    |
| Set.Creator            | Alt+Shift+S    | Alt+Shift+S    | Alt+Shift+S    | Alt+Shift+S    |

Table 9 Productivity! Pro Key Bindings for Emacs, Macintosh and Macintosh Code Warrior keymaps

| Tool                          | Emacs               | Mac                 | Mac CW              | Mac Safe            |
|-------------------------------|---------------------|---------------------|---------------------|---------------------|
| Assistants                    | Alt+Enter           | Alt+Enter           | Alt+Enter           | Alt+Enter           |
| Delegate.Insight              | Alt+Shift+M         | Alt+Shift+M         | Alt+Shift+M         | Alt+Shift+M         |
| Find.Matching.Brace           | Ctrl+\              | Ctrl+\              | Ctrl+\              | Ctrl+\              |
| Find.Matching.Code            | Ctrl+Shift+\        | Ctrl+Shift+\        | Ctrl+Shift+\        | Ctrl+Shift+\        |
| Introduce.Constructors        | Alt+Shift+C         | Alt+Shift+C         | Alt+Shift+C         | Alt+Shift+C         |
| Navigator.Insight             | Alt+Shift+N         | Alt+Shift+N         | Alt+Shift+N         | Alt+Shift+N         |
| Navigator.Next                | Ctrl+Page Down      | Alt+Page Down       | Ctrl+Page Down      | Ctrl+Page Down      |
| Navigator.Previous            | Ctrl+Page Up        | Alt+Page Up         | Ctrl+Page Up        | Ctrl+Page Up        |
| Persistent.Bookmarks.Navigate | Alt+Shift+B         | Alt+Shift+B         | Alt+Shift+B         | Alt+Shift+B         |
| Select.Class                  | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Select.CodeBlock              | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Select.Method                 | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Select.Statement              | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> | <i>Not Assigned</i> |
| Selection.Narrow              | Ctrl+Shift+W        | Ctrl+Shift+W        | Ctrl+Shift+W        | Ctrl+Shift+W        |
| Selection.Expand              | Ctrl+W              | Alt+Shift+W         | Ctrl+W              | Ctrl+W              |
| Smart.Clipboard.Insight       | Alt+Shift+V         | Alt+Shift+V         | Alt+Shift+V         | Alt+Shift+V         |
| Smart.Clipboard.PopPaste      | Ctrl+Alt+Insert     | Ctrl+Alt+Insert     | Ctrl+Alt+Insert     | Ctrl+Alt+Insert     |
| Smart.Clipboard.Swap          | Ctrl+Shift+Insert   | Ctrl+Shift+Insert   | Ctrl+Shift+Insert   | Ctrl+Shift+Insert   |
| Smart.Templates.Expand        | Ctrl+J              | Ctrl+J              | Ctrl+J              | Ctrl+J              |

|                                |                |                |                |                |
|--------------------------------|----------------|----------------|----------------|----------------|
| Smart.Templates.ExpandLast     | Ctrl+Shift+J   | Ctrl+Shift+J   | Ctrl+Shift+J   | Ctrl+Shift+J   |
| Smart.Templates.ExpandOnTheFly | Alt+Shift+J    | Alt+Shift+J    | Alt+Shift+J    | Alt+Shift+J    |
| Highlight Local References     | Ctrl+Alt+Enter | Ctrl+Alt+Enter | Ctrl+Alt+Enter | Ctrl+Alt+Enter |














## Productivity! Tools Icons

---

The following table shows icons used by tools included into Productivity!

**Table 10. Productivity! Tools Icons**

| Icon  | Description  |
|---|--|
|    | <a href="#">Browse.Insight</a> used for fast browsing classes using short class names  |
|    | <a href="#">Browse.Members</a> used for fast browsing declared members of a class  |
|    | <a href="#">Class.Insight</a> allows finding and inserting a class using a short class name  |
|    | <a href="#">Easy.JavaDoc</a> provides easy generation of JavaDoc for selected members  |
|    | <a href="#">Easy.JavaDoc.Insight</a> provides easy generation of JavaDoc for selected members  |
|    | <a href="#">Get.Creator</a> easy creation of getters   |
|    | <a href="#">GetSet.Creator</a> easy creation of getters and setters for selected fields  |
|    | <a href="#">Set.Creator</a> easy creation of setter methods  |
|    | <a href="#">Implement.Insight</a> used for fast interface implementing   |
|  | <a href="#">Smart.Instantiate</a> allows you to instantiate a class variable or even implement an anonymous class in seconds                 |
|  | <a href="#">Constructor.Insight</a> allows you to quickly create constructors  |
|  | <a href="#">Override.Insight</a> allows you to easily override methods   |
|  | <a href="#">Context.Insight</a> allows you to check the current context and navigate from there  |
|  | <a href="#">Help.Insight</a> allows easy viewing of help topics (if any) for an identifier within the current context in the cursor position |
|  | <a href="#">Help.Insight.OnMembers</a> allows easy viewing of help topics for a member within the current context in the cursor position     |
|  | <a href="#">Smart.Braces</a> options icon  |
|  | Cache Refresh Actions Group  |
|  | Full refresh of Productivity! classes cache  |
|  | Refresh cache for classes included into selected Refresh Groups  |
|  | Refresh cache for classes included into project libraries  |
|  | Refresh cache for classes included into project only   |

| Professional Edition Tools' Icons   |  |
|---|--|
|    | <a href="#">Delegate.Insight</a> provides an easy way to generate methods, which implementations are delegated to another object (delegate). |
|    | <a href="#">Introduce.Constructor</a> allows easy generation of constructors intended to initialize appropriate fields of the class.         |
|    | Expands <a href="#">Smart.Template</a> with the name corresponding the word at caret or invokes <a href="#">Smart.Templates.Insight</a> .    |
|    | Invokes <a href="#">Persistent.Bookmarks.Navigate</a> that allows navigation to desired bookmark.  |
|    | This tool allows viewing of local clipboard queue and pasting one or several selected fragments in the editor.                               |
|    | This action allows swapping the content of the clipboard with currently selected block of code.  |
|    | This action allows consecutive popping and pasting of code fragments from the local clipboard history in the LIFO order.                     |
|    | Invokes the <a href="#">Navigator.Insight</a> window that allows choosing an object to navigate.   |
|    | This action allows to expand current selection incrementally to outer source element   |
|  | This action allow to narrow current selection incrementally to inner source element.   |
|  | Shows the <a href="#">Task List</a> , which allows viewing and managing tasks.   |

## Known Issues and Limitations

---

1. Productivity! Classes Cache
  - Since only public classes may be cached, the tools that depend on the cache allow working with the public classes only.
  - Cache may not be automatically refreshed during adding and/or removing classes, packages, and libraries, as well as upon changes to the project class and source paths. In such cases you should refresh the class cache manually or schedule the refresh at the project make or build.
  - To avoid using the already cached classes that belong to the previously removed packages, you should refresh the cache for all classes.
2. The tools that operate with the words under cursor may sometimes improperly handle the words with underscores.
3. Productivity! shortcuts are designed and tested to eliminate any possible conflicts with the JBuilder shortcuts in any standard JBuilder keymap. However there remains a possibility of conflicts with some of JBuilder plug-ins, other applications and those functionality of the operational system that use the same shortcuts for other purposes.
4. Several different Insights may be shown simultaneously.
5. [Smart.Instantiate](#) always allows instantiation of classes that have constructors with the package access only, without checking the actual package.
6. Working with Inner Classes.
  - [Override.Insight](#) and [GetSet.Creator](#) are unable to place caret at the methods generated for anonymous inner classes or for inner classes defined in the methods. The caret position in this case will be unchanged.
  - [Override.Insight](#) and [GetSet.Creator](#) are unable to resolve the inner classes stated as super classes or super interfaces for any other inner class. Thus it is not possible to override methods or to generate access methods for the fields defined in such inner classes.
7. [Help.Insight](#).
  - Shortcuts to non-local HTML pages may not work for external browsers under Microsoft Windows 2000.
  - JTextPane used in [Help.Insight](#) may hang JBuilder when displaying huge HTML pages and/or jumping to non-existing anchors.
  - [Help.Insight](#) may find classes members if the documentation was generated in compliance with standard JavaDoc doclet only.
  - [Help.Insight](#) may show improper documentation page for java.io package.
8. External browser invocation works under Win32 platform only.

## Known Issues and Limitations

9. Resizing of the Insight popups may not work properly in some cases.
10. The Help button in the Insight popups may remain highlighted after invocation of JBuilder help viewer.

## Productivity! Feedback

---

As part of continuing efforts to improve our product, we welcome your comments, suggestions and general feedback on the project.

If you have questions about Productivity!, please feel free to contact us for further information at [productivity@jproductivity.com](mailto:productivity@jproductivity.com) or visit our site using the following URL: <http://www.jproductivity.com>.

If you discover any issues or defects in Productivity!, please send the description of them to [productivity@jproductivity.com](mailto:productivity@jproductivity.com). We'd appreciate if you could provide us additional information that may definitely help us to fix these problems:

1. JBuilder version.
2. Operational system version and vendor.
3. List of third-party open tools installed in your JBuilder.
4. Exceptions stack trace and any error output. You can see it if you run JBuilder along with console.
5. Running threads dump (it makes sense if JBuilder is not responding). You can see it if you run JBuilder along with console and press Ctrl+Break shortcut in the focused console