

***Productivity! 1.0***  
*for Borland JBuilder*

***User Manual***

Copyright 2000-2001 AMIS Software. 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 © 2000-2001 AMIS Software <http://www.softamis.com>

## Contents

---

<b>PRODUCTIVITY! OVERVIEW .....</b>	<b>8</b>
<b>INSTALLATION INSTRUCTIONS .....</b>	<b>9</b>
PRODUCTIVITY! KEY INSTALLATION .....	9
<i>How to Obtain Key File .....</i>	<i>9</i>
<i>How to Install Key File .....</i>	<i>9</i>
PRODUCTIVITY! HELP INSTALLATION .....	9
UNINSTALLING PRODUCTIVITY! .....	9
<b>PRODUCTIVITY! TOOLS .....</b>	<b>10</b>
COMMON INSIGHTS FEATURES .....	12
<i>Context Switching .....</i>	<i>12</i>
<i>Help Support .....</i>	<i>12</i>
CLASS.INSIGHT .....	13
<i>Class.Insight Actions .....</i>	<i>13</i>
<i>Showing Navigation Pane .....</i>	<i>14</i>
<i>Options Dependency .....</i>	<i>14</i>
BROWSE.INSIGHT .....	15
<i>Browse.Insight Actions .....</i>	<i>15</i>
<i>Options Dependency .....</i>	<i>16</i>
BROWSE.MEMBERS .....	16
HELP.INSIGHT .....	17
<i>Navigation Pane .....</i>	<i>17</i>
<i>Hyperlink.Help .....</i>	<i>17</i>
<i>Integration with Other Insights .....</i>	<i>18</i>
<i>Options Dependency .....</i>	<i>18</i>
IMPLEMENT.INSIGHT .....	19
<i>Code Changes Synchronization .....</i>	<i>20</i>
<i>Options Dependency .....</i>	<i>20</i>
OVERRIDE.INSIGHT AND CONSTRUCTOR.INSIGHT .....	20
<i>Code Changes Synchronization .....</i>	<i>21</i>
<i>Options Dependency .....</i>	<i>21</i>
CONTEXT.INSIGHT .....	21
<i>Options Dependency .....</i>	<i>22</i>
EASY.JAVADOC AND EASY.JAVADOC.INSIGHT .....	22
<i>Easy.JavaDoc .....</i>	<i>22</i>
<i>Easy.JavaDoc.Insight .....</i>	<i>23</i>
<i>Options Dependency .....</i>	<i>23</i>
IMPORTS.BEAUTIFY .....	24
<i>Options Dependency .....</i>	<i>25</i>
SMART.INSTANTIATE .....	25
<i>Showing Navigation Pane .....</i>	<i>26</i>
<i>Options Dependency .....</i>	<i>27</i>
HYPERLINK.NAVIGATE .....	27
<i>Options Dependency .....</i>	<i>27</i>
HYPERLINK.HELP .....	28
<i>Options Dependency .....</i>	<i>28</i>
GETSET.CREATOR .....	28
SMART.BRACES .....	29

Options Dependency.....	30
<b>PRODUCTIVITY! OPTIONS .....</b>	<b>31</b>
PROJECT PROPERTIES DIALOG .....	32
<i>General Page</i> .....	32
<i>Code Style Page</i> .....	34
<i>JavaDoc Page</i> .....	36
<i>Cache Page</i> .....	38
EDITOR OPTIONS DIALOG .....	41
<i>General Page</i> .....	41
<i>Usage Page</i> .....	45
<i>Delays Page</i> .....	47
<i>Smart.Braces Options (Editor Options)</i> .....	49
IDE OPTIONS DIALOG .....	52
<i>Productivity! Page</i> .....	52
<b>PRODUCTIVITY! KEY BINDINGS .....</b>	<b>54</b>
KEY BINDINGS FOR CUA, BRIEF AND VISUAL STUDIO KEYMAPS .....	55
KEY BINDINGS FOR EMACS, MACINTOSH AND MACINTOSH CODE WARRIOR KEYMAPS .....	56
<b>PRODUCTIVITY! ICONS .....</b>	<b>57</b>
<b>KNOWN ISSUES AND LIMITATIONS.....</b>	<b>59</b>
<b>PRODUCTIVITY! FEEDBACK .....</b>	<b>61</b>

## Tables

---

Table 1 Productivity! Tools .....	10
Table 2 Productivity! Key Bindings for CUA, Brief and Visual Studio keymaps.....	55
Table 3 Productivity! Key Key Bindings for Emacs, Macintosh and Macintosh Code Warrior keymaps.....	56
Table 4. Productivity! Icons .....	57

## Figures

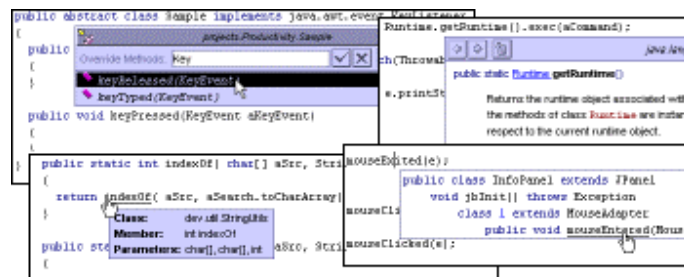
---

Figure 1 Class.Insight popup window .....	13
Figure 2 Browse.Insight popup window .....	15
Figure 3 Browse.Members popup window .....	16
Figure 4 Help.Insight popup window .....	17
Figure 5 Help.Insight popup invoked via Hyperlink.Help .....	18
Figure 6 JBuilder Member Insight with Help.Insight invoked .....	18
Figure 7 Implement.Insight popup window .....	19
Figure 8 Code generated by Implement.Insight.....	19
Figure 9 Override.Insight popup window .....	20
Figure 10 Context.Insight popup window.....	22
Figure 11 Easy.JavaDoc.Insight popup window .....	23
Figure 12 Import statements before Imports.Beautify invocation .....	24
Figure 13 Import statements after Imports.Beautify invocation .....	25
Figure 14 Smart.Instantiate popup window .....	26
Figure 15 Code before invocation of Smart.Instantiate.....	26
Figure 16 Code after invocation of Smart.Instantiate .....	26
Figure 17 Hyperlink.Navigate with hint that describes identifier under cursor .....	27
Figure 18 Hyperlink.Help popup window with help for identifier .....	28
Figure 19 GetSet.Creator popup window .....	29
Figure 20 Imports.Beautify options.....	32
Figure 21 Package Exclusion options.....	33
Figure 22 Methods Parameters Naming options.....	34
Figure 23 Fields Naming options .....	34
Figure 24 Generate throwing of UnsupportedOperationException options .....	35
Figure 25 General Code Style options .....	35

Figure 26 If JavaDoc exists options .....	36
Figure 27 Methods JavaDoc Generation options.....	37
Figure 28 Classes JavaDoc Generation options.....	37
Figure 29 JavaDoc Auto Generation options.....	38
Figure 30 Cache Auto Refresh options.....	39
Figure 31 Refresh Groups options.....	39
Figure 32 New/Edit Refresh Group Dialog .....	40
Figure 33 Refresh Now options .....	40
Figure 34 Import statements generation options .....	42
Figure 35 Search options .....	43
Figure 36 Sort Classes By options .....	44
Figure 37 Autocomplete options .....	44
Figure 38 Productivity! Insights Usage options.....	45
Figure 39 Invoke popups during debugging options .....	46
Figure 40 Help.Insight options .....	46
Figure 41 Highlight.Navigate options .....	47
Figure 42 Superclass Changing Policy options .....	47
Figure 43 Hyperlinks options .....	48
Figure 44 Help.Insight options .....	48
Figure 45 Context Discovering options .....	49
Figure 46 Smart.Braces options .....	49
Figure 47 Smart.Braces options (expanded) .....	50
Figure 48 Metal theme options.....	52
Figure 49 Default Steel Metal Theme sample .....	52
Figure 50 Plain Steel Theme sample .....	52

## 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:

- 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 and fields!
- 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!
- 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** to the *lib/ext* directory under your JBuilder installation.

**NOTE:** If you already have [Class.Insight](#) installed in your system, please remove it, as Productivity! has it already included. Also, you should remove the previous version of Productivity! (if any).

### Productivity! Key Installation

---

Productivity! requires a key file, which enables the Productivity! functionality.

#### How to Obtain Key File

---

In some cases, the evaluation key file **productivity.key** can be found in the downloaded archive. Otherwise, please visit [www.softamis.com](http://www.softamis.com) or contact [sales@softamis.com](mailto:sales@softamis.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 *.jbuilder4* (or *.jbuilder5*, 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 **productivity.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.

### Uninstalling Productivity!

---

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

## 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!

**Table 1 Productivity! Tools**

Tool	Description
<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="#">Browse.Insight</a>	<a href="#">Browse.Insight</a> allows quick finding Java classes with short names matching the word at the cursor position and browsing them or the appropriate help topics.
<a href="#">Browse.Members</a>	<a href="#">Browse.Members</a> allows quick finding members belonging to the current discovered context and browsing them.
<a href="#">Help.Insight</a>	<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.
<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.
<a href="#">Constructor.Insight</a>	<a href="#">Constructor.Insight</a> allows quick overriding class constructors.
<a href="#">Context.Insight</a>	<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.
<a href="#">Easy.JavaDoc</a>	<a href="#">Easy.JavaDoc</a> allows easy and convenient generating templates for JavaDoc comments for a particular method or class.
<a href="#">Imports.Beautify</a>	<a href="#">Imports.Beautify</a> is intended to arrange import statements into a clear and easy-readable form. It allows consolidating, alphabetical sorting and optional grouping all import statements for the current Java file.

## Productivity! Tools

<a href="#">Smart.Instantiate</a>	<a href="#">Smart.Instantiate</a> is an additional <a href="#">Class.Insight</a> functionality that allows adding instantiation of a particular class or interface.
<a href="#">Hyperlink.Navigate</a>	<a href="#">Hyperlink.Navigate</a> is a tool that allows easy and convenient navigation through symbols definitions basing on the concept of hyperlinks.
<a href="#">Hyperlink.Help</a>	<a href="#">Hyperlink.Help</a> allows easy and convenient viewing help topics for particular symbols.
<a href="#">GetSet.Creator</a>	<a href="#">GetSet.Creator</a> is a tool that allows easy creation of accessors and/or mutators for selected fields of a class.
<a href="#">Smart.Braces</a>	<a href="#">Smart.Braces</a> is a tool that allows easy creation of matching braces right while you are typing.

## 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.

## 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!

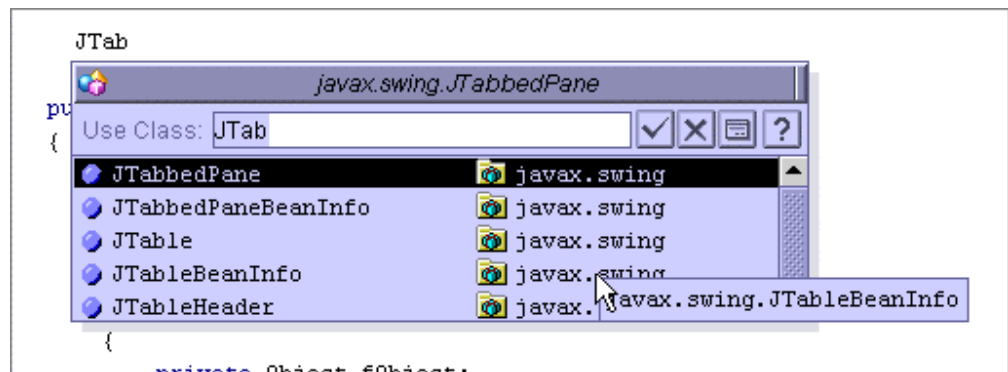


Figure 1 Class.Insight popup window

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.

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*

## 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.

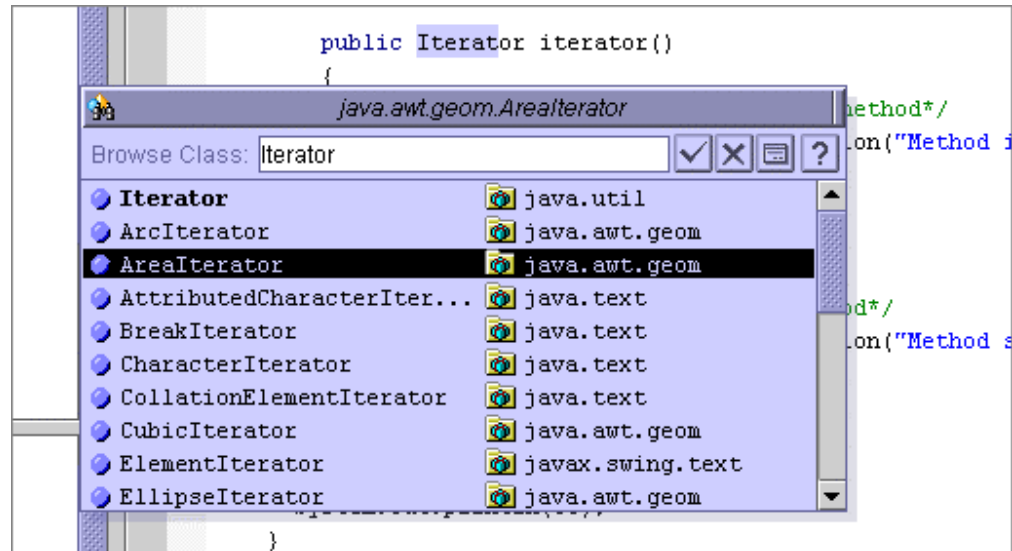


Figure 2 Browse.Insight popup window

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.

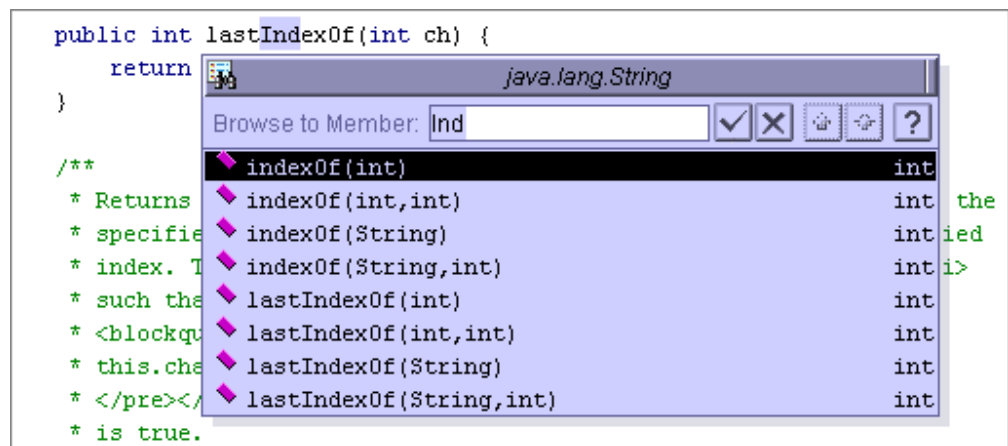


Figure 3 Browse.Members popup window

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.

## 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.

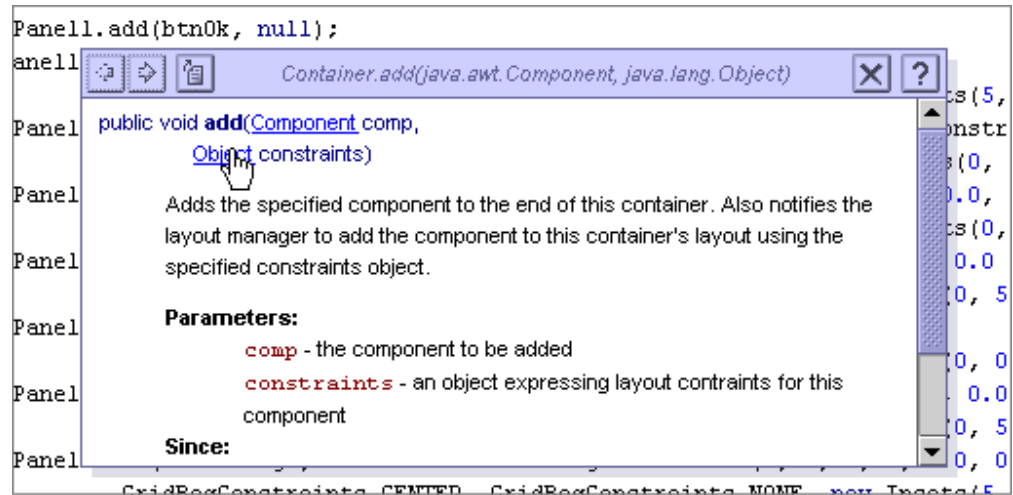


Figure 4 Help.Insight popup window

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.

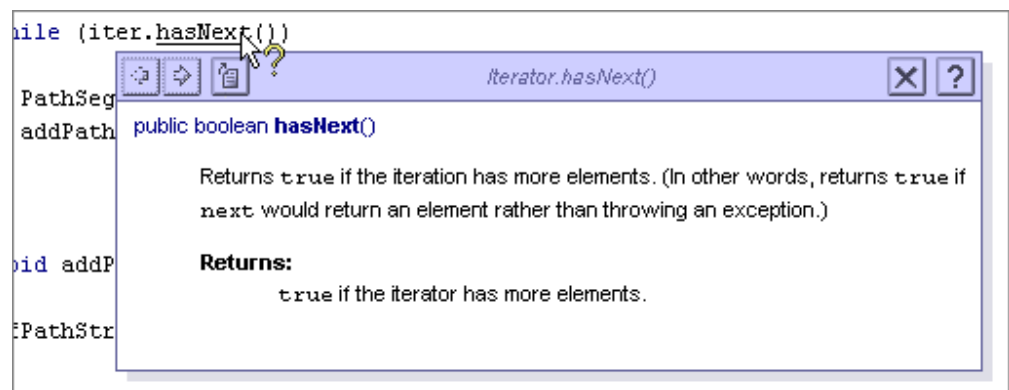


Figure 5 Help.Insight popup invoked via Hyperlink.Help

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 Member Insight and other Productivity! Insights. 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).

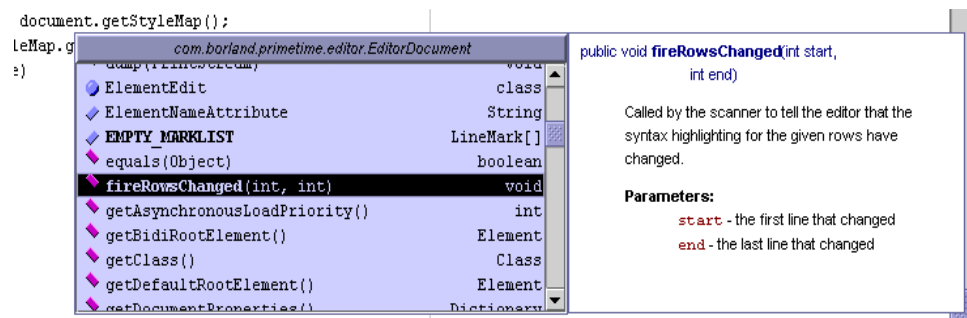


Figure 6 JBuilder Member Insight with Help.Insight invoked

### 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.

## 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.

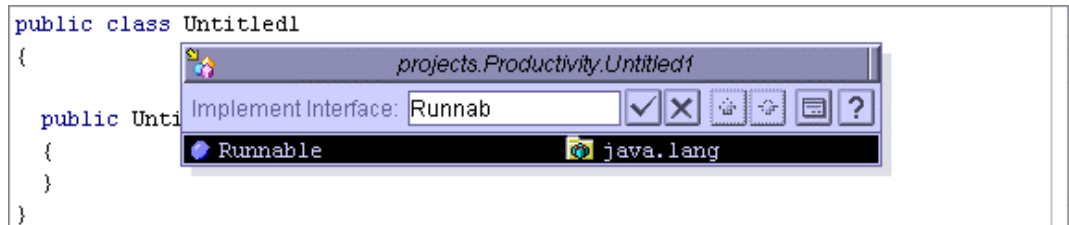


Figure 7 Implement.Insight popup window

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).

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

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

Figure 8 Code generated by Implement.Insight

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.

### 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.

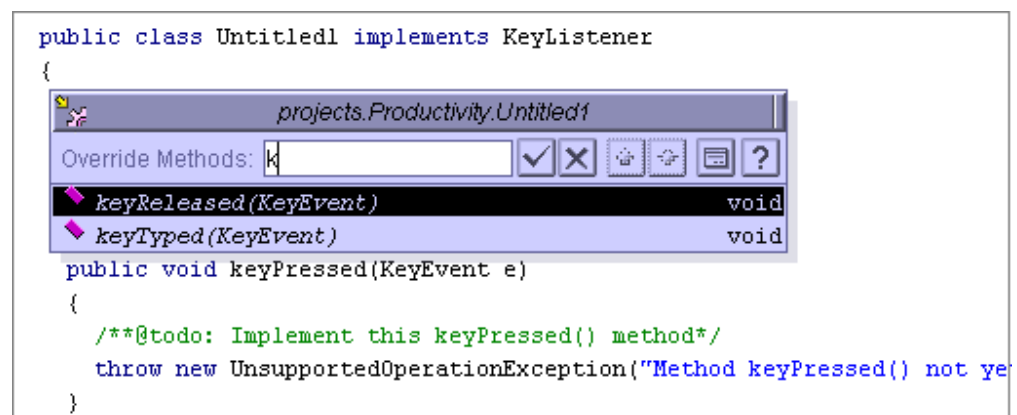


Figure 9 Override.Insight popup window

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*.

## 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.

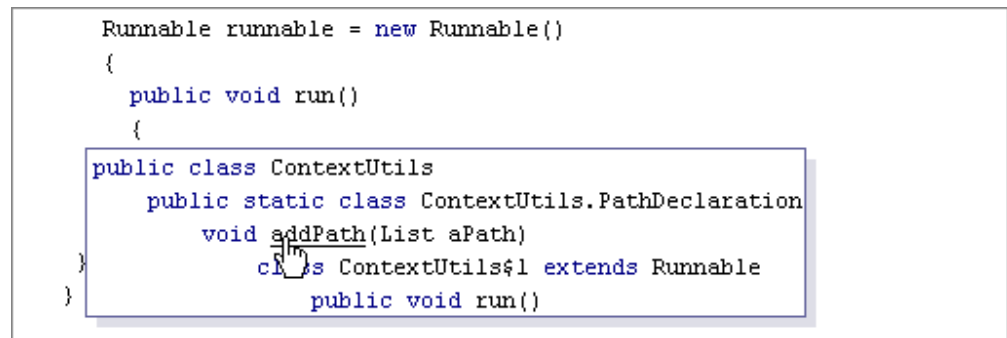


Figure 10 Context.Insight popup window

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.

## 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.

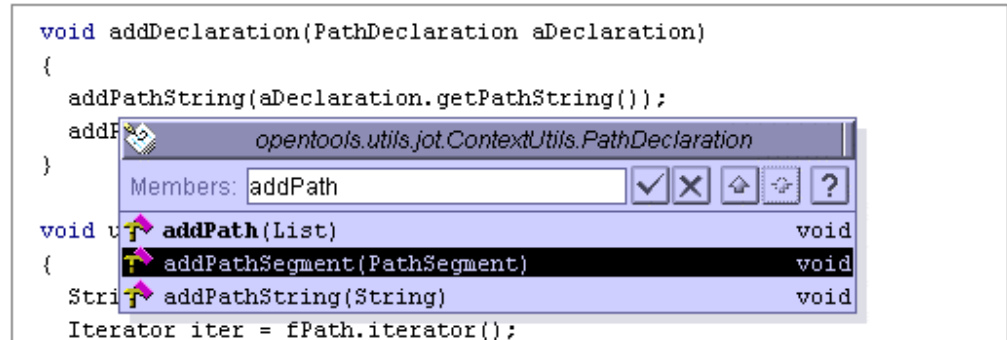


Figure 11 Easy.JavaDoc.Insight popup window

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.

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.

## **Imports.Beautify**

---

**Imports.Beautify** is intended to arrange import statements into a clear and easy-readable form. It allows consolidating, alphabetical sorting and optional grouping of all import statements in the current Java file.

Press **Ctrl + Alt + B** (CUA) to invoke it. In the following shots you can see imports before and after **Imports.Beautify** invocation.

```
import com.borland.primetime.node.Project;
import java.awt.Container;
import javax.swing.event.ChangeEvent;
import java.awt.event.KeyEvent;
import com.borland.jbuilder.jot.*;
import java.awt.event.ActionEvent;
import javax.swing.event.ListSelectionEvent;
import com.borland.primetime.node.Node;
import javax.swing.*;
import com.borland.jbuilder.node.JBProject;
import javax.swing.DefaultListModel;
import opentools.insight.ui.*;
import java.awt.Frame;
import com.borland.primetime.ide.StatusView;
import java.util.*;
import com.borland.jbuilder.node.JavaFileNode;
import com.borland.jbuilder.jot.ui.CodeStylePropertyGroup;
import javax.swing.text.*;
import javax.swing.tree.TreePath;
```

Figure 12 Import statements before Imports.Beautify invocation

```

import java.awt.Container;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.util.*;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ListSelectionEvent;
import javax.swing.text.*;
import javax.swing.tree.TreePath;

import com.borland.primetime.ide.StatusView;
import com.borland.primetime.node.Node;
import com.borland.primetime.node.Project;

import com.borland.jbuilder.jot.*;
import com.borland.jbuilder.jot.ui.CodeStylePropertyGroup;
import com.borland.jbuilder.node.JBProject;
import com.borland.jbuilder.node.JavaFileNode;

import opentools.insight.ui.*;

```

Figure 13 Import statements after Imports.Beautify invocation

### Options Dependency

---

**Imports.Beautify** optionally allows grouping imports by specified root packages so you can group your imports exactly as you want. To maintain grouping behavior you can use the **Imports.Beautify** option on the *Project Properties* | *Productivity!* | *General* and *Import Statements Generation* option the *Editor Options* | *Productivity!* | *General* property pages.

### 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 14 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.

A couple of samples of illustrating what [Smart.Instantiate](#) can do for you!



Figure 15 Code before invocation of Smart.Instantiate

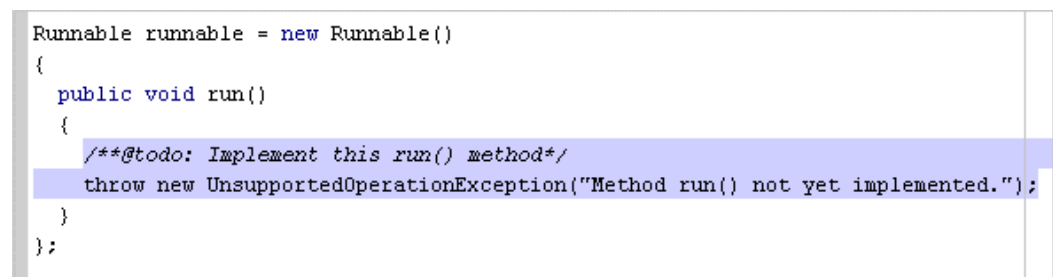


Figure 16 Code after invocation of Smart.Instantiate

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.

### 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.

## 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.

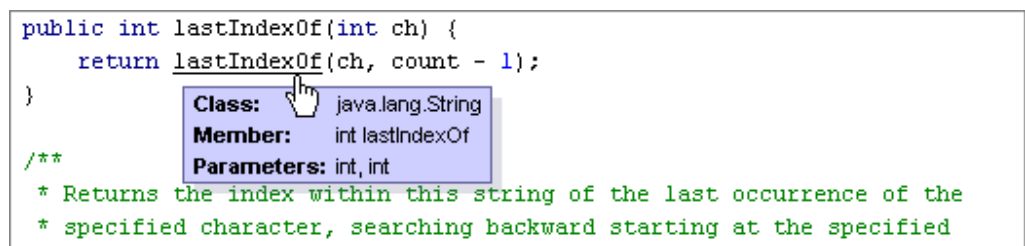


Figure 17 Hyperlink.Navigate with hint that describes identifier under cursor

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.

## Hyperlink.Help

---

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

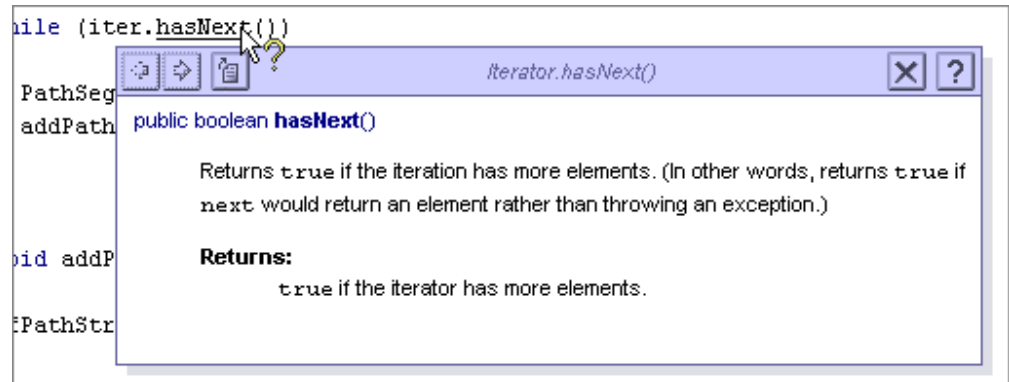


Figure 18 Hyperlink.Help popup window with help for identifier

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.

## 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.

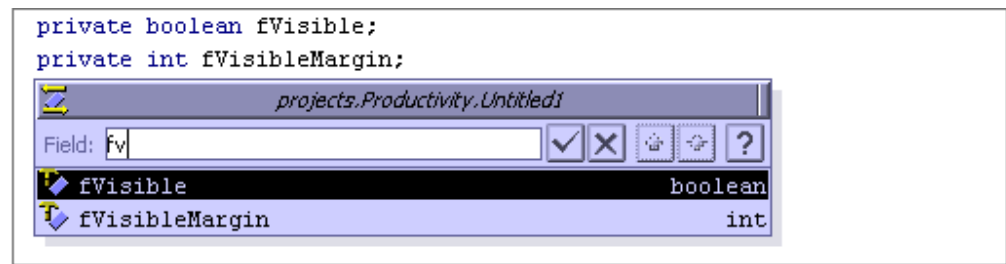


Figure 19 GetSet.Creator popup window

**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.

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.

## 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
```

## Productivity! Options

---

Productivity! 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

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.

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. *Imports.Beautify*
2. *Packages Exclusion*

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

---

### Imports...

The [Imports.Beautify](#) tool provides the ability to beautify imports by grouping import statements and sorting them within every group. This option allows you to customize [Imports.Beautify](#) functioning.

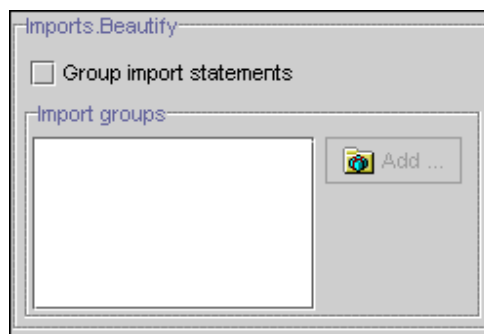


Figure 20 Imports.Beautify options

#### *Group import statements*

Please select this checkbox if you want to let the [Imports.Beautify](#) group import statements during imports beautifying. If you disable grouping, [Imports.Beautify](#)

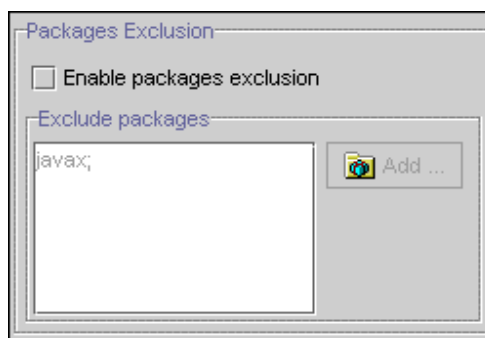
will only consolidate import statements according to options specified in *Editor Options | Productivity! | General | Import statements generation* and sort them.

#### *Import groups*

With the help of this panel, you can specify how to group import statements during import beautifying. Either type the root package of the group manually (only valid Java symbols are allowed, ';' separates packages), or add them using the *Select Package* dialog, invoked by the *Add...* button. Import statements will be grouped in the order specified by this option.

#### **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 21 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*
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.

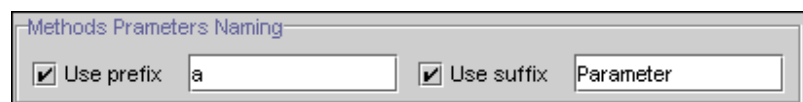


Figure 22 Methods Parameters Naming options

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.

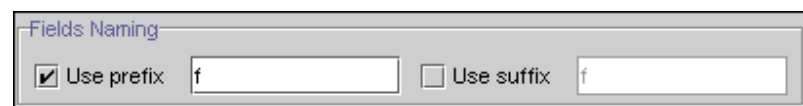


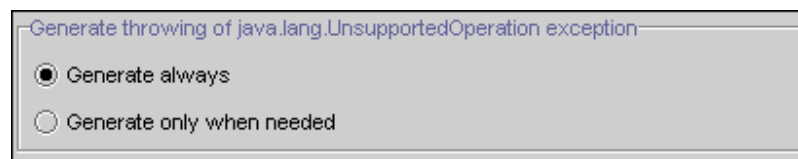
Figure 23 Fields Naming options

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.

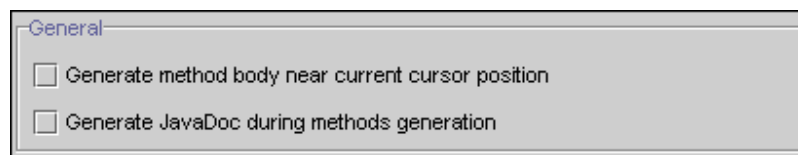


**Figure 24 Generate throwing of UnsupportedOperationException options**

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 25 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.

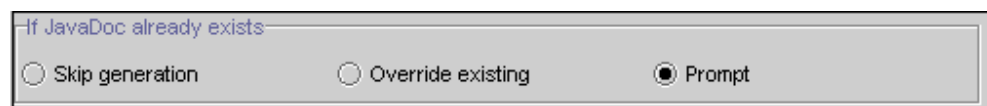


Figure 26 If JavaDoc exists options

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.

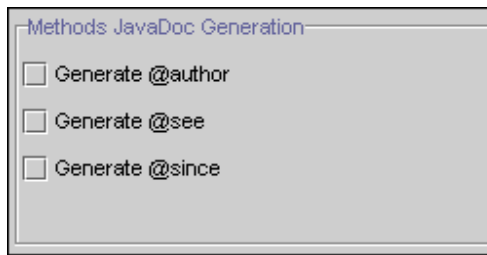


Figure 27 Methods JavaDoc Generation options

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.

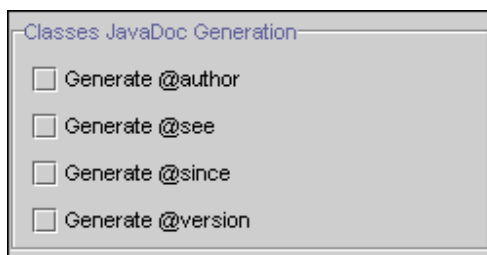


Figure 28 Classes JavaDoc Generation options

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.

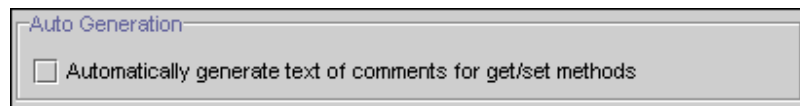


Figure 29 JavaDoc Auto Generation options

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.

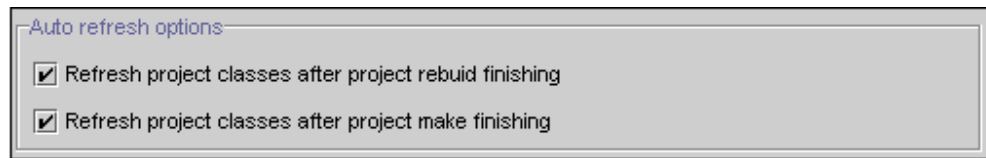


Figure 30 Cache Auto Refresh options

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.

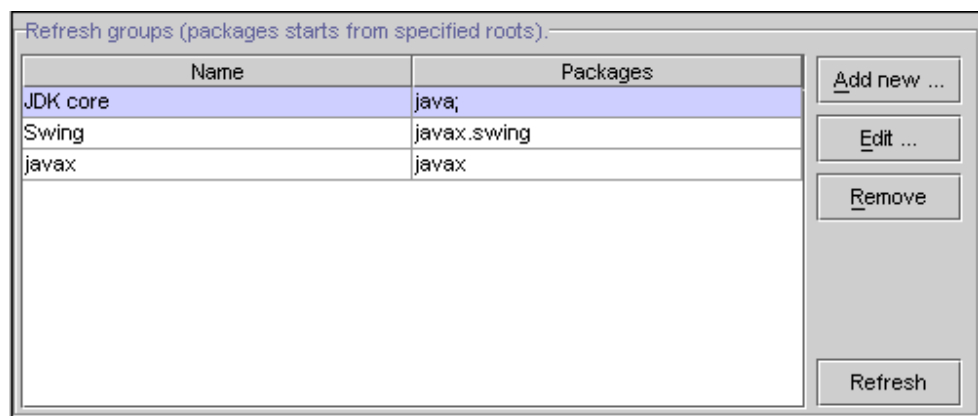


Figure 31 Refresh Groups options

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*.

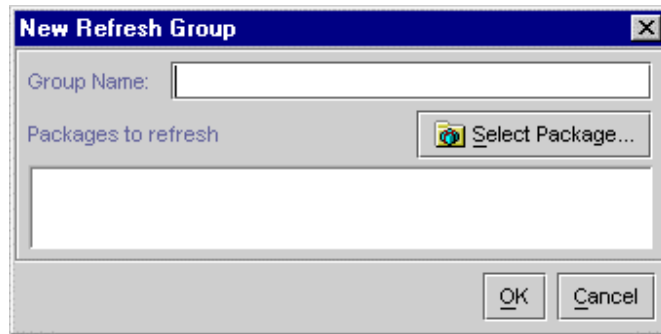


Figure 32 New/Edit Refresh Group Dialog

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.

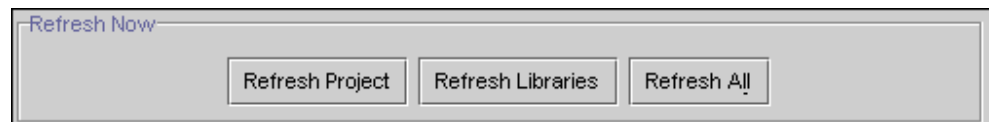


Figure 33 Refresh Now options

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).

## 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.

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.

### General Page

---

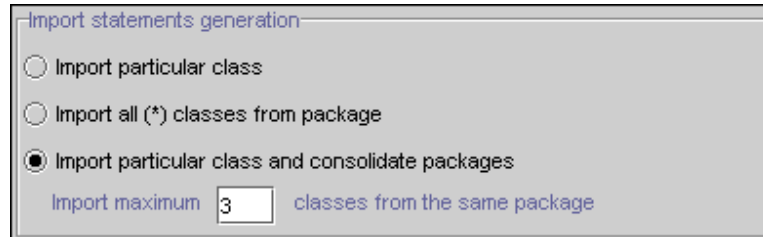
The *General* page of the Productivity! *Editor Options* page provides the following options:

- *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 34 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.

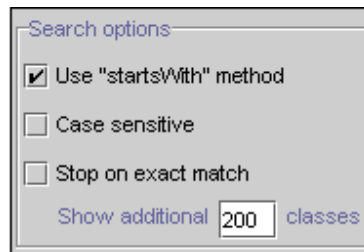
*Import particular class and consolidate packages*

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 35 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.

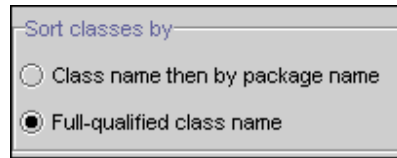
*Show additional classes*

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 36 Sort Classes By options**

The following options are available for classes sorting control:

*Class name then by package name*

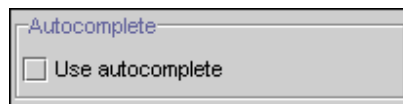
If you select this option, the classes will be sorted according to the class name and then class package;

*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 37 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.

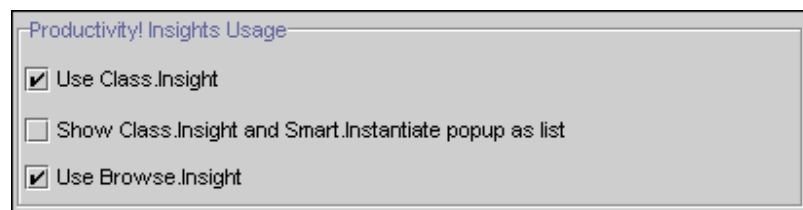


Figure 38 Productivity! Insights Usage options

#### *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.

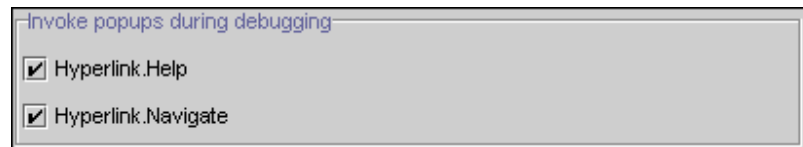


Figure 39 Invoke popups during debugging options

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.

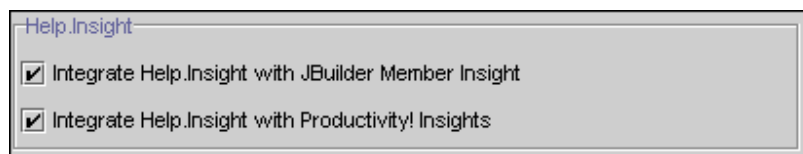


Figure 40 Help.Insight options

*Integrate Help.Insight with JBuilder Member Insight*

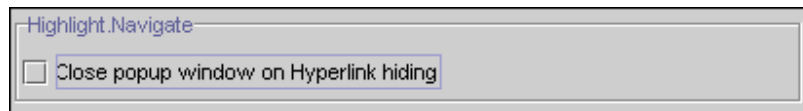
If this checkbox is selected, [Help.Insight](#) will be integrated with JBuilder Member Insight. In such mode, as soon as you change the selection within the Member Insight popup window list, [Help.Insight](#) with the help for a selected item will appear near the Member Insight popup window.

*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.

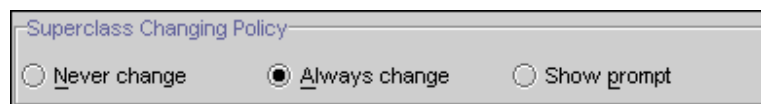


**Figure 41** Highlight.Navigate options

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 42** 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.

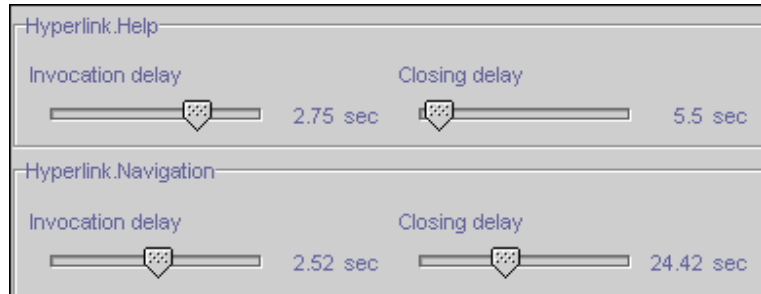
**Delays Page**

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**

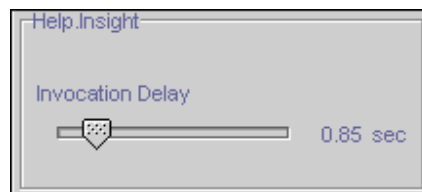
With these options you may specify the delays used for invocation and closing of popup windows displayed by the [Hyperlink.Help](#) and [Hyperlink.Navigate](#) tools.



**Figure 43 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 44 Help.Insight options**

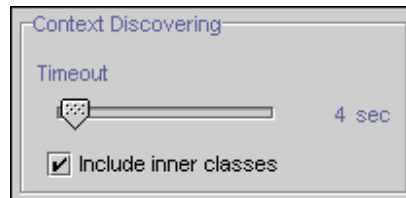
You may also indicate whether [Help.Insight](#) will be integrated with the JBuilder built-in insights using the *Enable Help for built-in JBuilder insights* checkbox. From the *Enable Help for Productivity! Insights* option you can specify whether or not [Help.Insight](#) will be integrated with the 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 45 Context Discovering options**

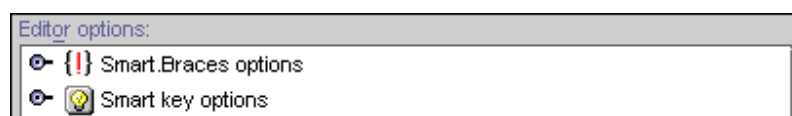
The reason for this is the limitation of JBuilder JOT subsystem that requires significant amount of time (up to tens of seconds) to retrieve information about the inner classes. To avoid hang-up of JBuilder, the time required for collection of context information was limited to 2 seconds. Thus, if JOT 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.

### **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 46 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.

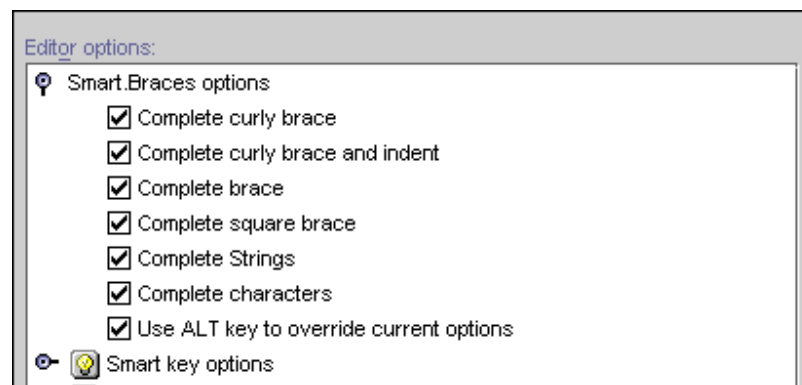


Figure 47 Smart.Braces options (expanded)

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 "

## Productivity! Options

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
```

## 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

### Productivity! Page

The Options page of the Productivity! IDE Options pages 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).

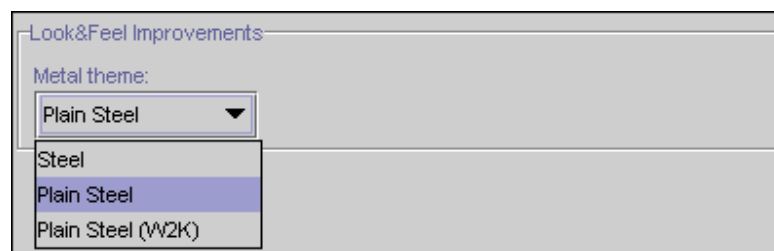


Figure 48 Metal theme options

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

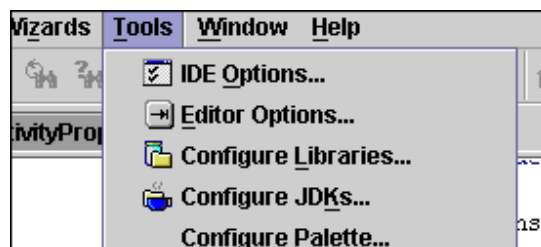


Figure 49 Default Steel Metal Theme sample

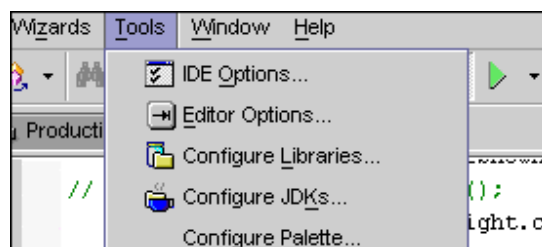


Figure 50 Plain Steel Theme sample

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

## **Productivity! Options**

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.

## 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)

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

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

## 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 3 Productivity! Key Key Bindings for Emacs, Macintosh and Macintosh Code Warrior keymaps**

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

## Productivity! Icons




---

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

**Table 4. Productivity! 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="#">Imports.Beautify</a> action allows you to consolidate, group and sort your import statements
	<a href="#">Smart.Braces</a> options icon
	Cache Refresh Actions Group
	Full refresh of Productivity! classes cache

## Productivity! Icons

	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

## 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.

## Known Issues and Limitations

- [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.
  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@softamis.com](mailto:productivity@softamis.com) or visit our site using the following URL: <http://www.softamis.com>.

If you discover any issues or defects in Productivity!, please send the description of them to [productivity@softamis.com](mailto:productivity@softamis.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