By Nick Decker
Engineering Supervisor

BDT Tips for Less Pain and More Gain

The CodeEditor in the BDT (Business BASIC Development Tools) Eclipse plug-in is a powerful, full-featured editor for BBj® programs. What makes the BDT one of the best options for developing BBj programs is that it not only inherits a number of features from the DLTK (Dynamic Languages Toolkit) framework on which it is built, but BASIS engineers enhanced it further by adding several supplementary capabilities specific to the BBj language. As with most sufficiently advanced editors, it's capable of so many functions that often times new users are not aware of all that si avaialble. Worse yet, Business BASIC developers may not be using the CodeEditor to its fullest, and may be missing out on capabilities, time saving tips, and other productivity enhancements that 'power users' put into practice on a daily basis. This article does not attempt to cover the myriad of options that the BDT offers, but rather covers a few of the 'can't live without' features that we have compiled after polling several BASIS engineers who spend a good chunk of their day inside the Eclipse IDE.

Quick Access to Anything

Eclipse can do innumerable things without a mouse like using only keystrokes to access menu options, tool buttons, and more. Many options and commands are available that, in all likelihood, most programmers don't know about or wouldn't know how to get to. For example, press [Ctrl]+[Shift]+**L**, or [Cmd]+[Shift]+**L** (⌘⌥**L**) on OS X, to see the myriad of key commands that are not only available, but customizable to your liking. Finding the right keystroke or command may seem like looking for a needle in a haystack, but Eclipse offers a great way to give you easy access to whatever you're looking for with the Quick Access search option. Located at the top right of the toolbar, you can type in **bb** as shown in **Figure 1** to display a list of commands, menu items, preferences, and more that contain the letters **bb**.

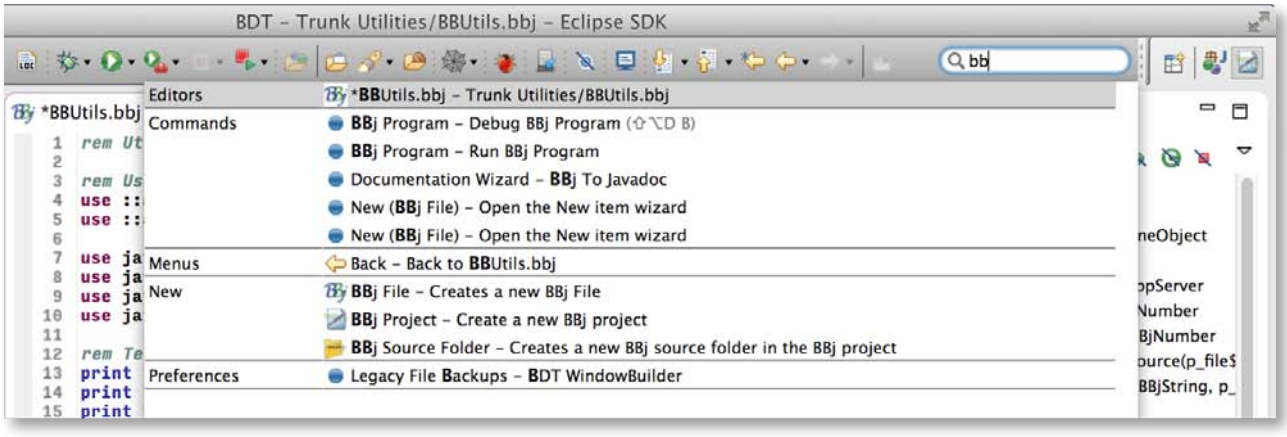


Figure 1. Fast access to almost everything is available from the Quick Access search

In practice, this Quick Access search feature is a huge timesaver. For example, you can type the word **font** into the search bar (shown in **Figure 2**), then select the first 'Commands' option to display the 'Fonts and Colors' preferences dialog. You can access the desired preference pane in just a few keystrokes using the Quick Access search, which is much faster than using the mouse to navigate the menu and preferences hierarchy, even assuming you know where the command resides in the first place.

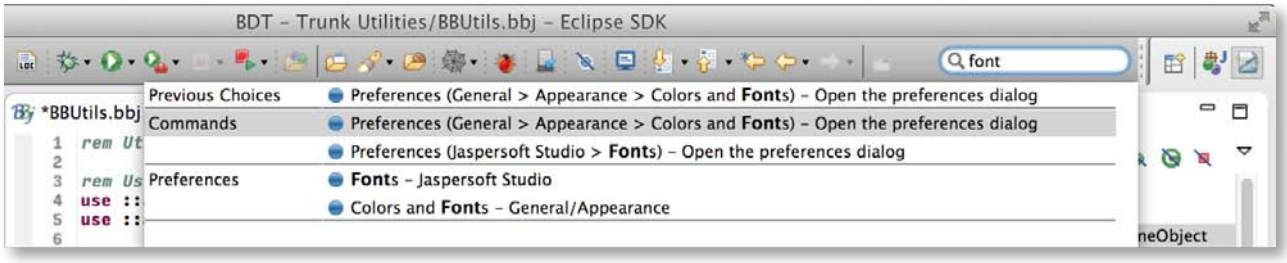


Figure 2. Quick access to the Colors and Fonts preference is just a few keystrokes away

Program Navigation – Outline View

Eclipse's 'Outline' view is at the top of most programmers' list of favorites because it provides an efficient way to navigate through a program. In a standard BBj program, it lists variable declarations, program labels, and functions. In an object-oriented program, it's even more useful as you can drill down into classes to see methods as well as class and method field variables. Everything has a unique icon shape and color so you can readily tell at a glance the difference between classes and interfaces, as well as private, protected, and public methods and variables.

The tree node entries are chock full of information too. For example, instead of just listing all possible methods in a class, the entries display each method's parameter list of variables and their type, and the return type of the method. **Figure 3** demonstrates this with the applyCss() method that appears highlighted in the outline view and selected in the CodeEditor.

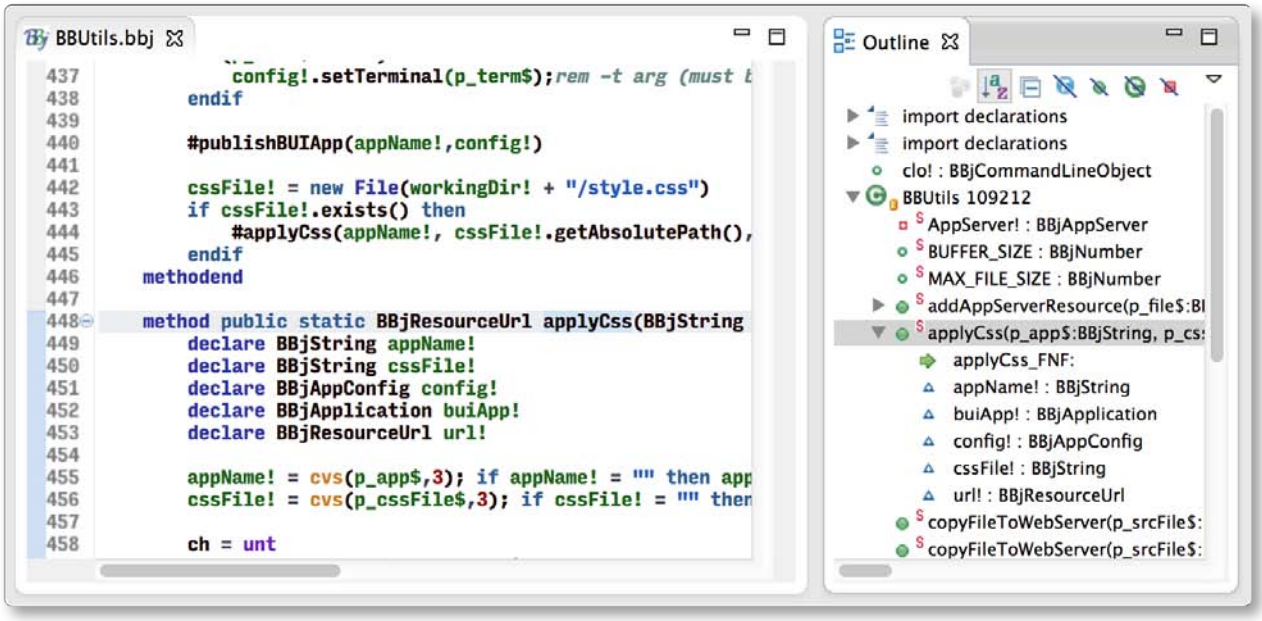
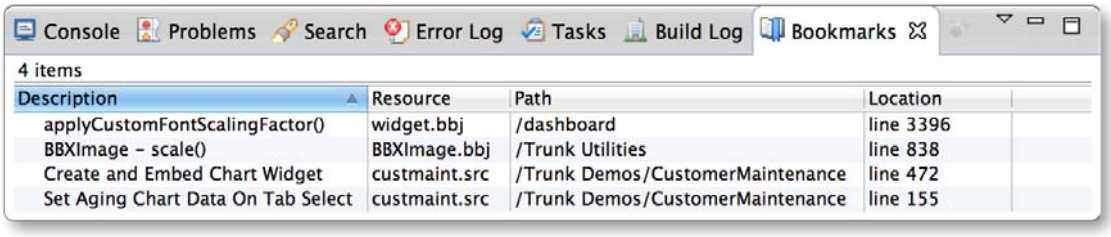


Figure 3. Navigating to a method in the CodeEditor (left) using the Outline view (right)

The Outline view's toolbar icons located at the top of the view make it even more flexible, allowing you to sort the view contents alphabetically, hide/show components, expand/collapse nodes, and more.

Program Navigation – Bookmarks

Bookmarks are another great way to quickly navigate through code and find often-used methods or routines. Simply select one or more lines of pertinent code, then select the [Add Bookmark…] option from the 'Edit' menu, and enter a meaningful name in the prompt. All of your saved bookmarks will appear in the 'Bookmarks' view as shown in **Figure 4**. If you do not see the bookmark view by default, go to the menu and click on Window > Show View > Other > Bookmarks, then place the view anywhere you desire in the IDE for quick reference.



Description	Resource	Path	Location
applyCustomFontScalingFactor()	widget.bbj	/dashboard	line 3396
BBXImage - scale()	BBXImage.bbj	/Trunk Utilities	line 838
Create and Embed Chart Widget	custmaint.src	/Trunk Demos/CustomerMaintenance	line 472
Set Aging Chart Data On Tab Select	custmaint.src	/Trunk Demos/CustomerMaintenance	line 155

Figure 4. The bookmarks view showing saved bookmarks from multiple programs

The bookmarks are associated with the program and selected code, so opening up a bookmark from the list causes the BDT to load the appropriate program, jump to the code of interest, and even select the original lines of code. You can freely add or remove code in the original program and the bookmark will still be valid, as Eclipse updates its location dynamically to stay in sync with any insertions or deletions.

Editing – Multiline Comments

The BBj language uses the REM Verb to indicate comments in a program, but what about the scenario when you would like to comment out a large block of code? Programmers often write test code intended to replace sections of their original program, but using the REM Verb to remark individual lines can be burdensome. The CodeEditor has an elegant solution to this, as you can highlight any number of lines of code, then select Source > Toggle Comment or the appropriate hotkey: [Ctrl]+/ in Windows and Linux or [Cmd]+/ (⌘/) on OS X. This instantly adds or removes REMs to every selected line, making it quick and easy to remark multiple lines of code at once. **Figure 5** shows the same block of code before and after toggling comments in the editor.

```
136
137     sliderFontSize!.setMinimum(25)
138     sliderFontSize!.setMaximum(150)
139     sliderFontSize!.setOpaque(0)
140     sliderFontSize!.setPaintLabels(1)
141     sliderFontSize!.setMajorTickSpacing(25)
142     sliderFontSize!.setMinorTickSpacing(5)
143     sliderFontSize!.setPaintTicks(1)
144     sliderFontSize!.setValue(100)
145
```

```
136
137 REM sliderFontSize!.setMinimum(25)
138 REM sliderFontSize!.setMaximum(150)
139 REM sliderFontSize!.setOpaque(0)
140 REM sliderFontSize!.setPaintLabels(1)
141 REM sliderFontSize!.setMajorTickSpacing(25)
142 REM sliderFontSize!.setMinorTickSpacing(5)
143 REM sliderFontSize!.setPaintTicks(1)
144 REM sliderFontSize!.setValue(100)
145
```

Figure 5. A block of code before (left) and after (right) toggling comments

Editing – Multiline Editing

Block selection mode is another sometimes little known favorite that can really be a time saver. You can simply toggle block selection mode using the keystrokes [Alt]+[Shift]+A on Windows and Linux or [Cmd]+[Opt]+A (⌘⌥A) on OS X. Turning on this editing mode allows you to select rectangular blocks of text rather than wrapped lines to easily delete, move, copy, or paste these blocks just as you would any other selection. After turning on block mode the font may change, depending on your configuration for the 'Text Editor Block Selection Font', indicating the new selection mode. You can then make rectangular selections (see **Figure 6**) where the selection is a one-character column from lines 412 to 419.

```
411   sql$ = "SELECT cast(invoice_date as SQL_CHAR) as invdate, sum(total_sales) as TOTAL_SALES "
412   "FROM ART03 "
413   "WHERE "
414   "firm_id = '01' "
415   "and "
416   "(invoice_date between '" + inputDStart!.getText() + "' AND '" + inputDFinish!.getText() + "') "
417   "and "
418   "slspsn_code = '" + salesperson!.get(salesperson$) + "' "
419   "GROUP BY invoice_date "
420
```

Figure 6. Selecting a column of text in block mode

After selecting the column, you can type in text that is then inserted on all eight of the selected lines simultaneously. **Figure 7** shows how we inserted the **`'sql$ = sql$ + '`** text on all of the lines at once, completing the code and resolving the syntax errors indicated in **Figure 6.**

```
411     sql$ = "SELECT cast(invoice_date as SQL_CHAR) as invdate, sum(total_sales) as TOTAL_SALES "
412     sql$ = sql$ + 'FROM ART03 "
413     sql$ = sql$ + 'WHERE "
414     sql$ = sql$ + 'firm_id = '01' "
415     sql$ = sql$ + 'and "
416     sql$ = sql$ + '(invoice_date between '" + inputDStart!.getText() + "' AND '" + inputDFinish!.getT
417     sql$ = sql$ + 'and "
418     sql$ = sql$ + 'slspsn_code = '" + salesperson!.get(salesperson$) + "' "
419     sql$ = sql$ + 'GROUP BY invoice_date "
420
```

Figure 7. Inserting text into eight lines simultaneously

Editing – Templates Do Your Typing

Developers usually find themselves typing certain code patterns frequently, such as the familiar FOR/NEXT verb loop or IF/THEN/ELSE conditional statements. Templates in Eclipse are similar to word processor macros, as they transform a few characters into predefined blocks of code. This not only saves time, but the resultant code is already indented and formatted, thus improving consistency. The BDT comes with several default templates that you can access via the Content Assist keystroke combination of [Ctrl]+[Space]. For example, type **`if`** and invoke Content Assist, after which you can select the desired built-in template. **Figure 8** shows what this looks like in the editor.

Highlighting a template in the popup window results in a preview of the defined code block, and selecting the template inserts the code into our program. Templates also save time by eliminating our need to position the cursor with the mouse. In the example above, after inserting the code into our program, the template selects the **`condition`** text. That means we can type in the appropriate condition for our program, which overwrites the placeholder text. Additionally, pressing the [Return] key causes the template to advance the cursor intelligently in between the IF/ENDIF lines. Not only did the template leave the THEN portion intact, it also correctly indented the cursor.

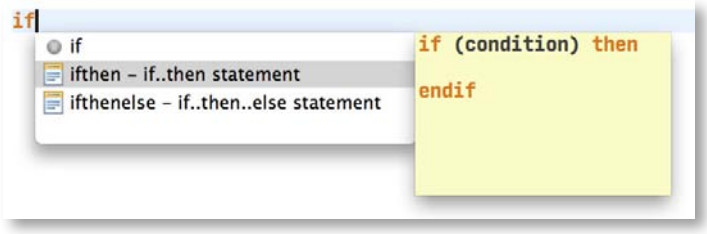


Figure 8. Using a template to insert a block of code

To get the most out of templates, create your own to help automate your development process. Templates can take advantage of pre-defined variables such as **`${cursor}`**, which specifies the cursor position when exiting edit mode, or **`${word_selection}`** that resolves to the content of the currently selected text. These variables allow you to add a level of sophistication and flexibility to your templates so that they will be more valuable than simply pasting in static blocks of code.

Editing – Revision Comparisons

On occasion, every programmer has broken a piece of code that used to work perfectly. Usually this occurs after hours of work and many changes to the code, making it a challenge to know which change is responsible for the problem. If you're using a source code repository, you can use Eclipse to compare your current code to the latest version in the repository. But what about when you're working on code that you haven't checked in yet? Eclipse handles that case too, as you can right-click on the program, select Compare With > Local History to open the 'History' view. Eclipse keeps a running list of revisions that are timestamped and available in the History view so that you can see a list of your recent changes and compare the current code to one of those versions. You can even configure the Local History to set limits such as retention time, saved changes per file, and maximum file size in your workspace preferences.

Editing – Quick Diffs

Side-by-side comparisons are definitely useful, but if you want a quicker, easier way to see the changes you've made in your code, use the real-time Quick Diff. You can configure Quick Diff in preferences and set the various colors for changes, additions, and deletions as well as configuring the reference source. You can set the reference source to 'Version on Disk', but comparing local copies to the latest in a source code repository yields much more functionality. After configuring it to show in the ruler, both the left and right edges of the CodeEditor will display colored blocks, visually depicting changes to the

code. In **Figure 9**, red corresponds to code deletions, green to additions, and blue to changes in the existing code. The left ruler shows changes for each adjacent line and the right ruler shows changes for the program as a whole. More specifically,

- Red number 1 indicates a place where code has been deleted, and after hovering your mouse over that block, a popup appears showing what the deleted code block looked like. If you want to undelete it, just copy it from the popup and paste it back into the program, or right click and select [Restore Deleted Lines].
- Green number 2 shows that this section of the code is new compared to what is checked in to the repository.
- Blue number 3 shows other places in the program where code has been changed.

Hovering over a block on the right ruler in **Figure 9** shows information about the section, and notes in a popup in the lower right that there is a new line label and it added 22 lines. Clicking on any block in the right hand ruler jumps immediately to that section of the program and selects the new or modified code. In this way, the Quick Diff not only shows you changes to your code at a glance, but it also serves as a rapid way to navigate in your program and jump to other modifications in the file.

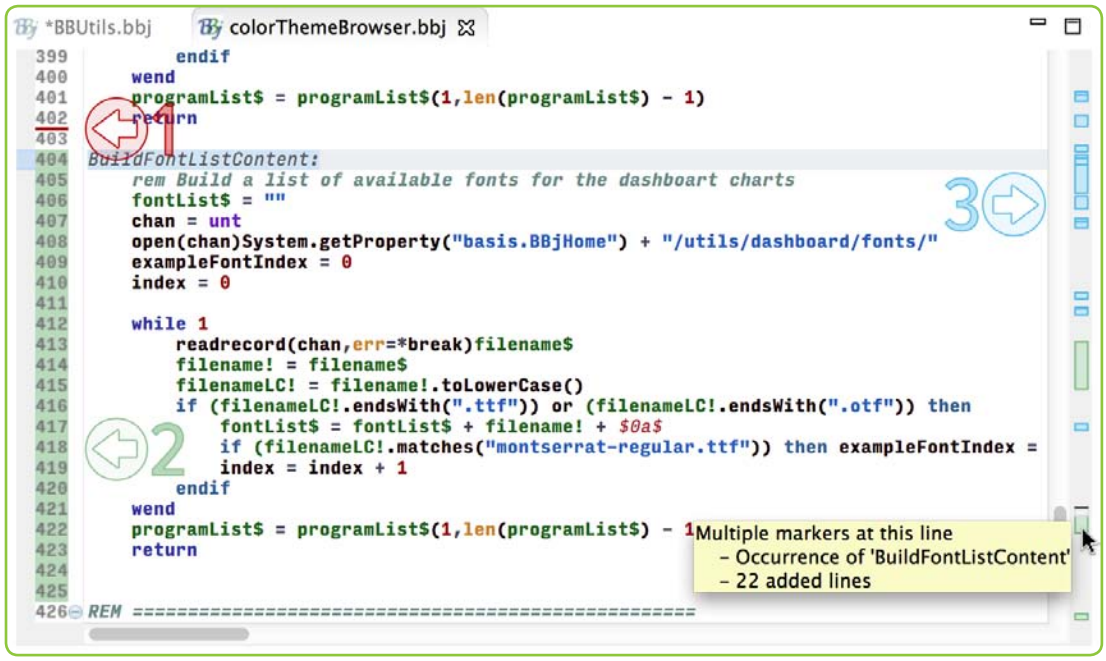


Figure 9. The Quick Diff ruler entries showing changes to the program

Memory Management

Eclipse is a multi-purpose development environment and you may have dozens of different modules besides the BDT installed to work on multiple projects, each with different editors and plug-ins. Memory usage can have a direct impact on performance and Eclipse gives you a way to keep an eye on memory consumption. You can mark the General > 'Show heap status' checkbox option in Eclipse's preferences to display information about the current Java heap usage. The heap status is located in Eclipse's bottom status bar, as shown in **Figure 10**. The graph and numbers indicate how much memory Eclipse is currently using versus how much memory you have allocated to it. It also includes a garbage can icon that when clicked performs a garbage collection to free up any reclaimable memory.

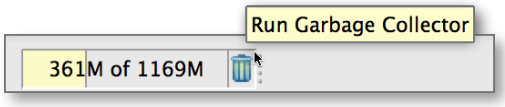


Figure 10. The amount of currently used versus allocated memory in Eclipse

If you find Eclipse constantly running close to its maximum heap allowance, you can increase its heap size as detailed in the FAQ *How do I increase the heap size available to Eclipse?* Depending on your usage of the IDE and your machine's amount of physical RAM, increasing the amount of memory available to Eclipse can dramatically improve performance.

Summary

Eclipse is such a powerful and capable development environment that detailing every aspect and feature would take an entire book (of which there are plenty, not coincidentally). This article took a different approach, and limited its scope to a select set of features that BASIS engineers use on a daily basis. If you have not heard of some of them, or have yet to put them into your development cycle, give them a try today and see how they can increase your productivity. ■

- Read these *BASIS International Advantage* articles:
  - *Have it Your Way With New BDT Preferences*
  - *Eclipse: The Toolset of the Future*
  - *Double Your Pleasure With Eclipse Plug-in Documentation*
- Check out these additional resources:
  - BASIS documentation
  - Eclipse documentation
  - FAQ *How do I increase the heap size available to Eclipse?*