Transitioning from NetBeans to the BDT IDE

by Jerry Karasz

Are you a developer trying to deliver improved software with additional functionality, maybe even under shorter deadlines? Are you tasked to turn out new releases that are robust and stable on ever-shortening project schedules? These types of everyday stresses make it more and more worth your time to explore every opportunity for increased productivity. Better documentation, training, new tools, and time-saving utilities can help make you more productive and more successful in writing, testing, and deploying solid programs. After all, who doesn't want to turn out more work better and faster? One offering that could help make this dream a reality is the Business BASIC Development Toolkit (BDT) Interactive Development Environment (IDE) that is available as a set of plug-ins for Eclipse. With a number of productivity-enhancing features designed to help you deliver better software more quickly, BDT may be just what you need. But the biggest roadblock may be the transition, as there is bound to be a learning curve associated with any new tool. After all, you already know how to use the NetBeans BASIS IDE and you're comfortable with it. So, how can you determine what benefits you can take advantage of by moving up to the BDT IDE in Eclipse?

This article provides an overview for developers like you who want to know what BDT has to offer and how to leverage its advanced features as quickly as possible. It also provides suggestions and ideas for transitioning an existing BBj® program from the NetBeans IDE to the BDT IDE by walking through this process using a sample BBj program. The issues and tradeoffs involved will become obvious as you follow an example BBj program all of the way from its NetBeans existence until it becomes a new BDT project that can be built, run, and updated. Since the most recent NetBeans IDE was released as part of BBj 14.22, and BDT is now up to

version 18.20, we will use these development environments to demonstrate the transition. If you have been working with an earlier BBj/NetBeans IDE environment, you may see some minor differences from what we present here, but any differences should not be significant. To read the full tutorial detailing this transition, see the Transitioning from NetBeans to the Eclipse IDE Guide.

To keep this simple, we'll use a demo program called "Reports" that is installed with both BBj 14.22 and BBj 18.xx (and which should also be available in future BBj releases). This is a representative BBj program that contains a number of interesting file types, but isn't so large that it is too difficult to understand. BBj installs the example source code for the Reports demo in **`<bbjhome>\demos\Reports\`**, where **`<bbjhome>`** is the location where you installed BBj. We will transition a copy of the Reports demo that we have in **`C:\Reports\`**, treating this code as though it was the source for your program.

The NetBeans BASIS IDE

To transition a BBj program from the NetBeans IDE to the BDT IDE, let's assume that you already have a computer set up and configured with the NetBeans IDE and that you have a program you developed there. Our example folder **`C:\Reports\`** appears in NetBeans as seen in **Figure 1**.

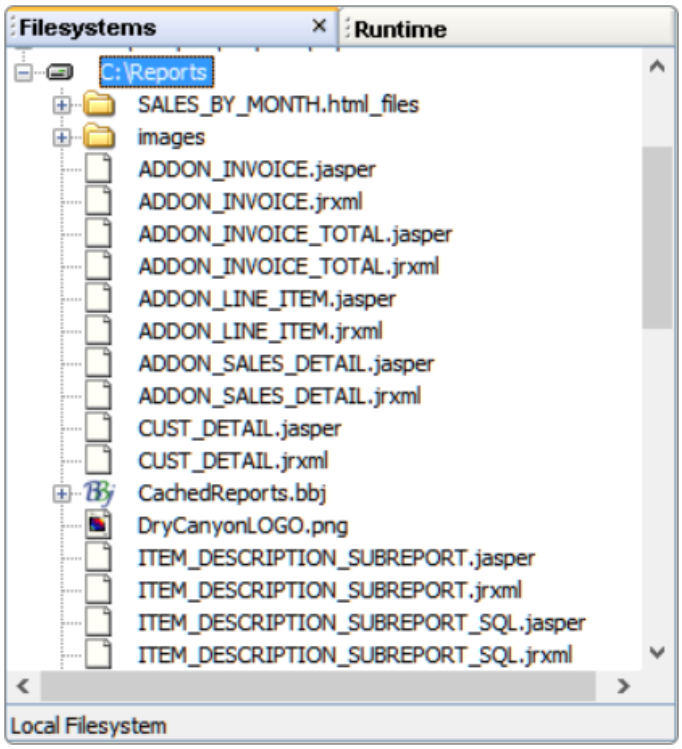

Figure 1. NetBeans Filesystems

Setting Up Eclipse and the BDT IDE

One note of caution: install Eclipse and the BDT plug-ins on a separate computer from the NetBeans IDE because the two IDEs require different versions of Java and different versions of BBj. The NetBeans IDE requires Java 7 and BBj 14.22, while the BDT IDE requires Java 8 or newer, and a newer BBj. The two Java/BBj combinations are incompatible.

Follow the instructions on Preparing Eclipse for BASIS-Provided Plug-ins to install Eclipse and the BDT plug-ins on a computer with Java 8 or newer. When choosing which URL to use for the BDT plug-ins, we used **`http://plugins.basis.com/bdt/18xx`** from the Eclipse Plug-ins page. To make sure that your setup is complete and correct, follow the instructions on Creating Your First BBj Project and create a BBj Project in BDT. Once you are successful, you can begin transitioning your existing program.

Transitioning the BBj Program

All of the source code for your program must be available to the BDT development environment as a completely separate copy (not shared). For simplicity, we used the most common case where your source code is stored on the NetBeans IDE development computer (in isolated local storage). We copied the source code to hard disk on the BDT IDE computer via a USB flash drive.

Once you create a new empty BBj Project named Reports, it is ready for the source code. Import the code using BDT's Import File System option by right-clicking the Reports project and selecting **Import...** Then expand the Reports project in the BDT Explorer. You should now see all of the files from the Reports demo in the BDT Explorer as shown in **Figure 2**. Remember this is a copy of the source, not the original which remains on the NetBeans IDE computer.

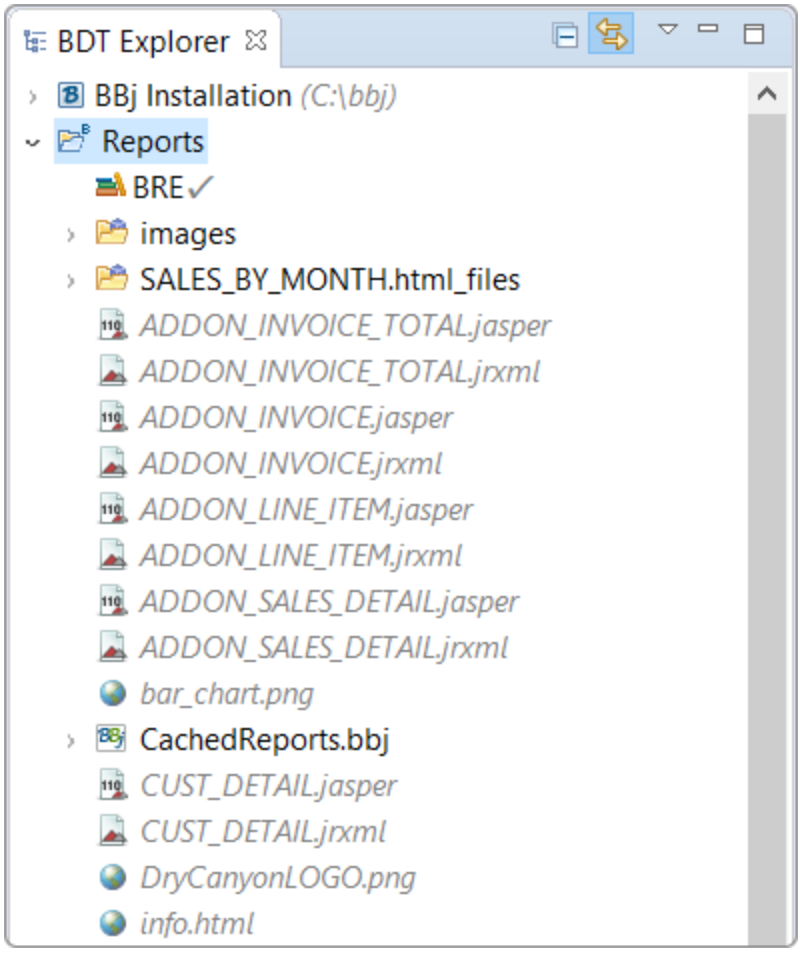

Figure 2. The Imported Reports Code in BDT's BBj Project

Now that our BBj program source code is in a BBj Project in the BDT IDE, we are ready to work with it there. There are a number of common commands we can execute on our Reports BBj Project such as edit, save, compile/tokenize, debug, execute as a GUI program, and execute as a BUI program.

Remember that the BDT Perspective and the BDT Explorer view are key. Most of BDT's functionality is only available when using these Eclipse tools, and you learned about them earlier in the Creating Your First BBj Project document.

BDT offers three different ways to launch a BBj program: executing it as a Graphical User Interface (GUI) program, executing it as a GUI program in the debugger, or executing it as a Browser User Interface (BUI) program. For simplicity, we will demonstrate launching our program as a GUI program. For more details on launch options, see the full Transitioning from NetBeans to the Eclipse IDE Guide.

To launch our BBj program as a GUI application from the BDT Explorer, expand the Reports project and right-click the **`Reports.src`** file. Then, select **Run As > BBj Program** (see **Figure 3**).

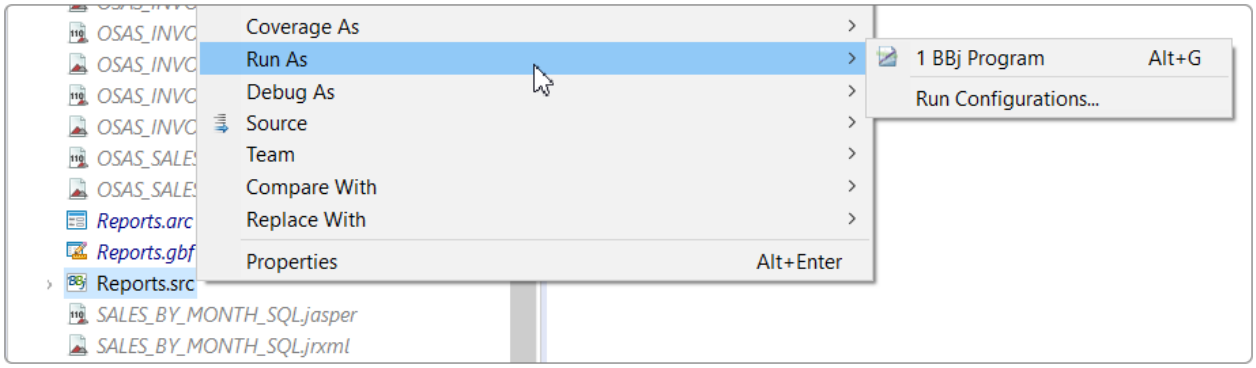

Figure 3. Running **`Reports.src`** as a BBj Program

BDT builds the project and then opens a BBj SysConsole window and a BBj Reports window similar to **Figure 4** (this may take a moment or two).

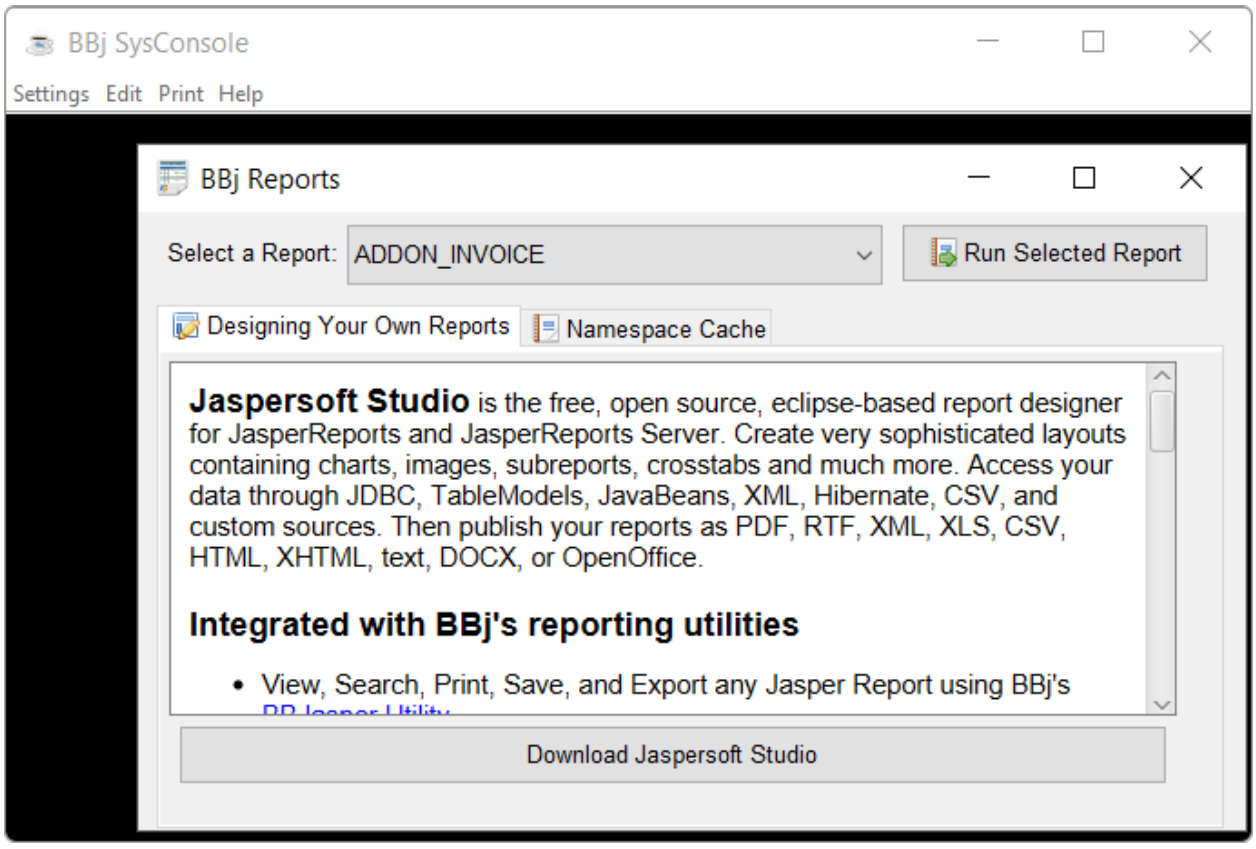

Figure 4. The **`Reports.src`** Windows

When you are satisfied that it is running normally, close both displays.

Now let's talk about editing BBj projects in BDT. You can edit at the project level (add a new project, edit the properties of an existing project, or add a new file or folder to an existing project - whether in the project root folder or in a sub-folder). You can also edit at the file level (add content to an existing file, edit the properties of an existing file, or delete an existing file). We can't demonstrate every edit option here, but let's demonstrate that our new BBj Project is valid by doing three simple editing examples: a BBj source file, an ARC resource file, and a GBF event file.

Editing a BBj Source File

Let's edit the **`Reports.src`** file to demonstrate how to make and view code changes. But doing so carries with it a **warning:** **`Reports.src`** is a generated file, created for the Reports demo when AppBuilder built the **`Reports.gbf`** file. Any changes we make to this copy of **`Reports.src`** will be overwritten the next time AppBuilder builds **`Resource.gbf`**. In a real-world environment, use extreme caution editing BBj source files created by AppBuilder; instead, you should edit **`Reports.gbf`** and build the GBF file, renaming the resulting file as desired.

For the purposes of this demonstration, we don't care if our changes are overwritten later, so we'll go ahead and edit it in a BBj CodeEditor. **Figure 5** displays the CodeEditor for **`Reports.src`**.

Reports.src

```
1  rem ' Generated by ProcessEventsBuilder (November 11, 2015 at 13:12:54)
2  rem ' To modify this program, edit the .gbf project file in AppBuilder.
3
4  gb__sysgui$ = "X0"
5  gb__sysgui = unt
6  open (gb__sysgui)gb__sysgui$
7  gb__sysgui! = bbjapi().getSysGui()
8  dim gb__event$:tmpl(gb__sysgui)
9  gb__handle = resopen("Reports.arc")
10 gosub gb__init_win_101
11 gosub Init
12
13 process_events
14
15 gb__init_win_101:
16 gb__win_101! = gb__sysgui!.createTopLevelWindow(gb__handle,101)
17 gb__win_101!.getControl(0).setCallback(gb__sysgui!.ON_WINDOW_MOVE,"W101_C0_WIN_MOVE
18 gb__win_101!.getControl(0).setCallback(gb__sysgui!.ON_RESIZE,"W101_C0_WIN_RESIZE")
19 gb__win_101!.getControl(0).setCallback(gb__sysgui!.ON_CLOSE,"W101_C0_WIN_CLOSE")
20 gb__win_101!.getControl(103).setCallback(gb__sysgui!.ON_BUTTON_PUSH,"W101_C103_PUSH
21 gb__win_101!.getControl(104).setCallback(gb__sysgui!.ON_BUTTON_PUSH,"W101_C104_PUSH
22 gb__win_101!.getControl(105).getControl(101).setCallback(gb__sysgui!.ON_BUTTON_PUSH
23 gb__win_101!.getControl(107).getControl(102).setCallback(gb__sysgui!.ON_BUTTON_PUSH
24 gb__win_101!.getControl(107).getControl(104).setCallback(gb__sysgui!.ON_BUTTON_PUSH
25 gb__win_101!.getControl(107).getControl(105).setCallback(gb__sysgui!.ON_LIST_CHANGE
26 gb__win_101!.getControl(107).getControl(108).setCallback(gb__sysgui!.ON_EDIT_MODIFY
27 gb__win_101!.getControl(107).getControl(110).setCallback(gb__sysgui!.ON_EDIT_MODIFY
28 return
29
30 W101_C0_WIN_MOVE:
```

Figure 5. **`Reports.src`** in a BBj CodeEditor

Right after line 16 that contains the following code:

```
gb__win_101! = gb__sysgui!.createTopLevelWindow(gb__handle,101)
```

insert a new line with the following code:

```
gb__win_101!.setTitle("New BBj Reports Title")
```

This will override the previous window title so that we can tell that our change is used. Save this change and click the [Run As…] toolbar button (see **Figure 6**) to run our modified **`Reports.src`** program.

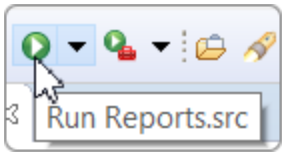

Figure 6. The 'Run' Toolbar Button for **`Reports.src`**

You should see the GUI program run showing the new title, "New BBj Reports Title", indicating that our changes were applied and run. Once this happens, you can dismiss the BBj Reports application and either revert your changes or leave the modified code, as you like.

Editing an ARC Resource File

Let's edit our **`Reports.arc`** (ARC resource) file to demonstrate how to use WindowBuilder. In the BDT Explorer view, double-click the **`Reports.arc`** file. to open a WindowBuilder ARC Editor. Click "101 NamespaceCacheWindow", and BDT graphically displays that TLC (ChildWindow) as captured in **Figure 7**.

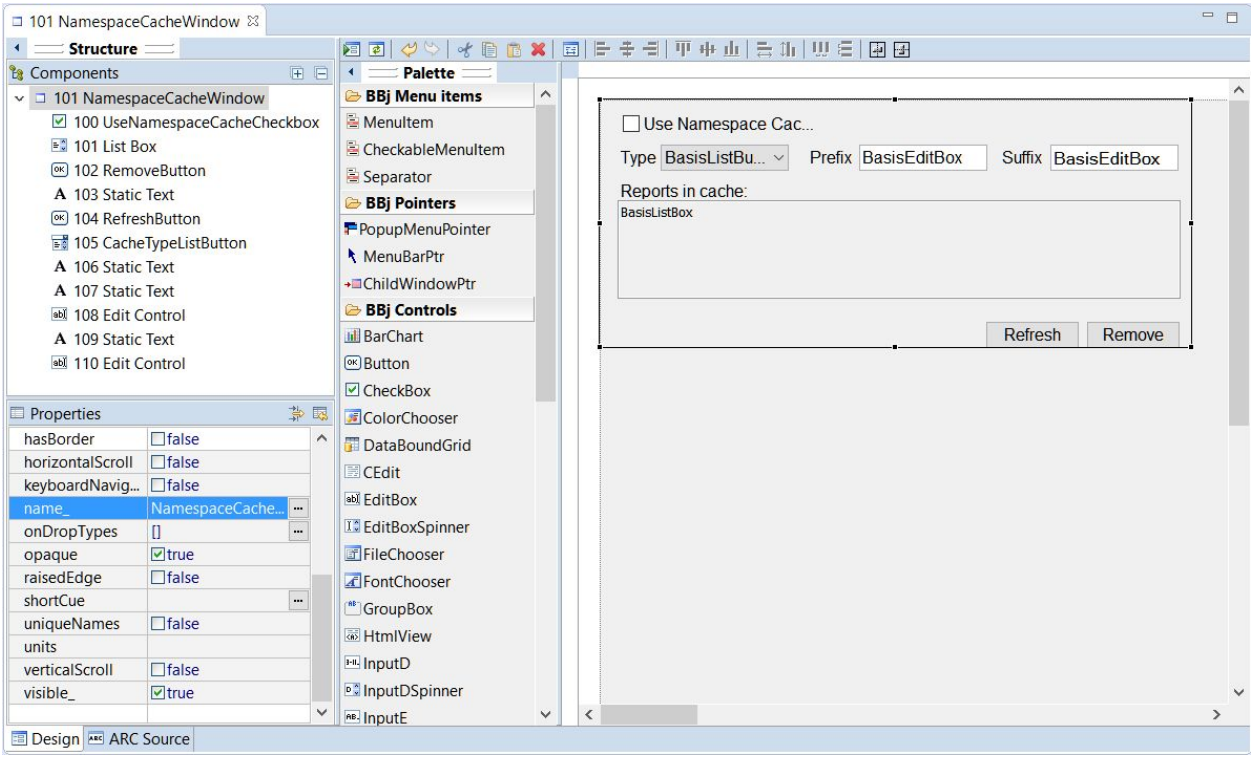

Figure 7. A Child Window from **`Reports.arc`** in a WindowBuilder ARC Editor

Select the ARC Source tab at the bottom of the WindowBuilder ARC Editor at any time to view the read-only ARC text that will be written out to the .arc file when you save (see **Figure 8**).

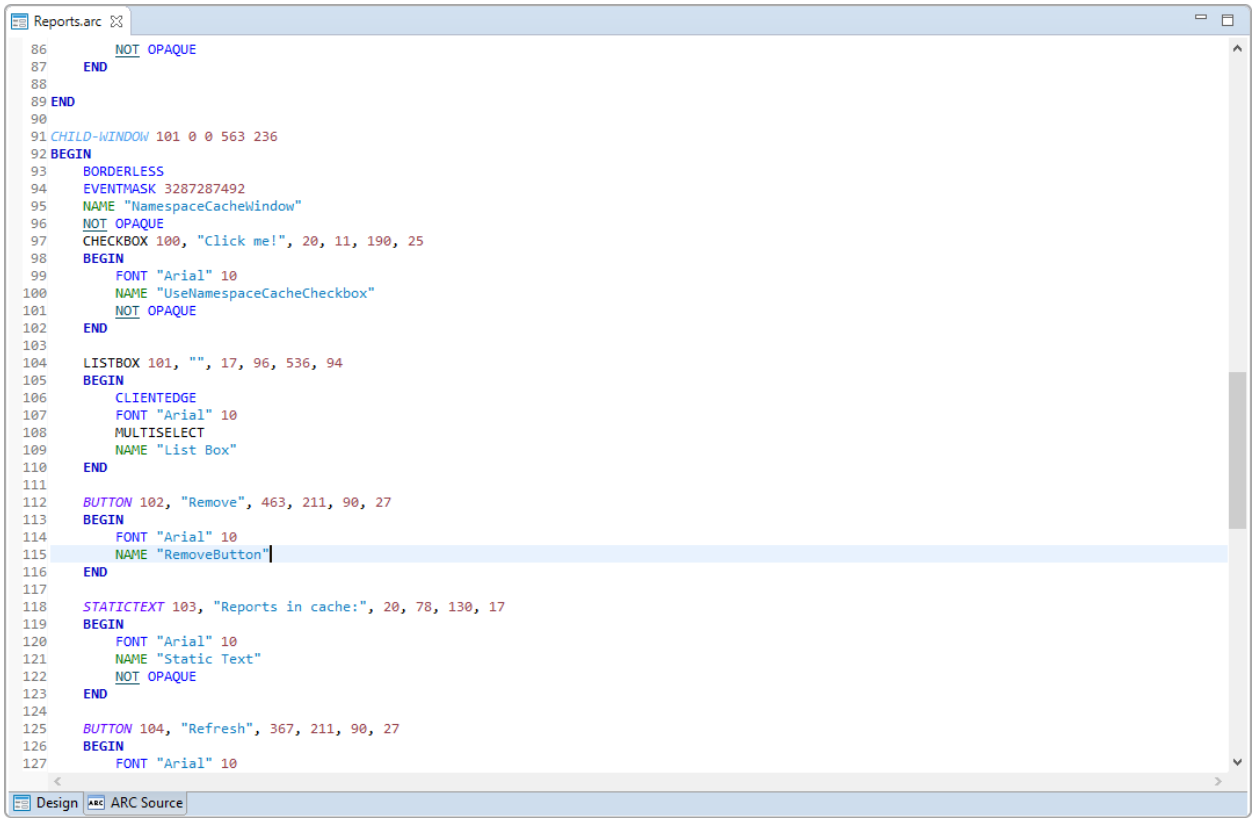

Figure 8. The ARC Source Tab Showing a Child Window's ARC Text

Back in the Design tab, select "100 UseNamespaceCacheCheckbox" in the Components pane. Notice that the Properties display now shows the properties for the checkbox. Edit the Text property there and change the text to "Check Me!" as shown in **Figure 9**.

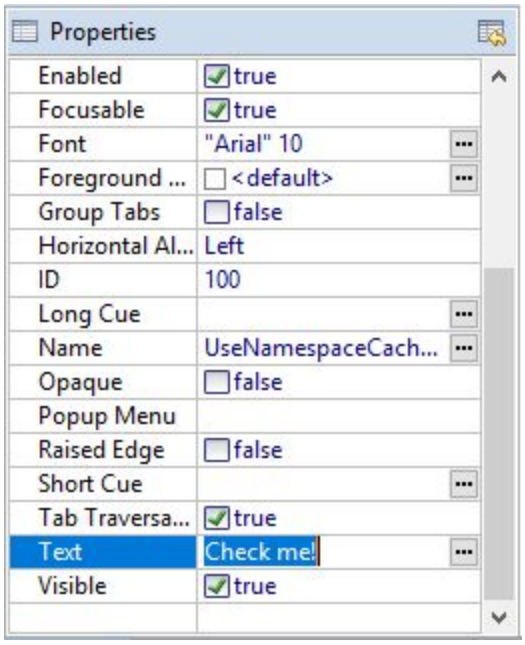

Figure 9. Editing the 'Text' Property

Save this change and click the [Run As…] toolbar button to run our modified **`Reports.src`** code. Select the Namespace Cache tab, and examine the checkbox in the upper left of the tab. Notice that it now shows "Check Me!" instead of "Use Namespace Cache" (**Figure 10**).

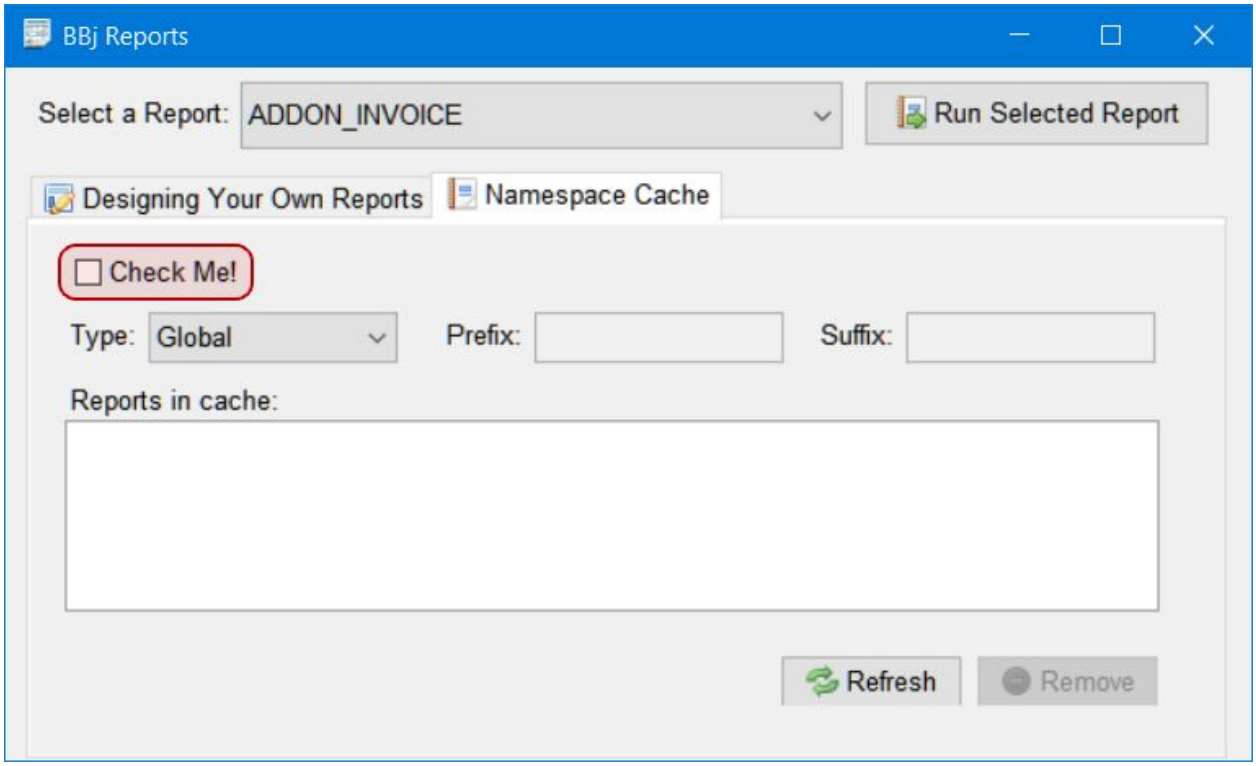

Figure 10. Viewing Our Edited CheckBox Text

Dismiss the BBj Reports application, and either revert your change to the checkbox or leave it, as you like.

Editing a GBF Event File

Finally, let's edit our **`Reports.gbf`** (AppBuilder GBF event) file to demonstrate how to use AppBuilder. We'll change the event behavior when the user clicks the [Refresh] button on the Use Namespace Cache tab. In the BDT Explorer view, double-click the **`Reports.gbf`** entry to open an AppBuilder GBF editor that displays the configuration for that file as shown in **Figure 11**.

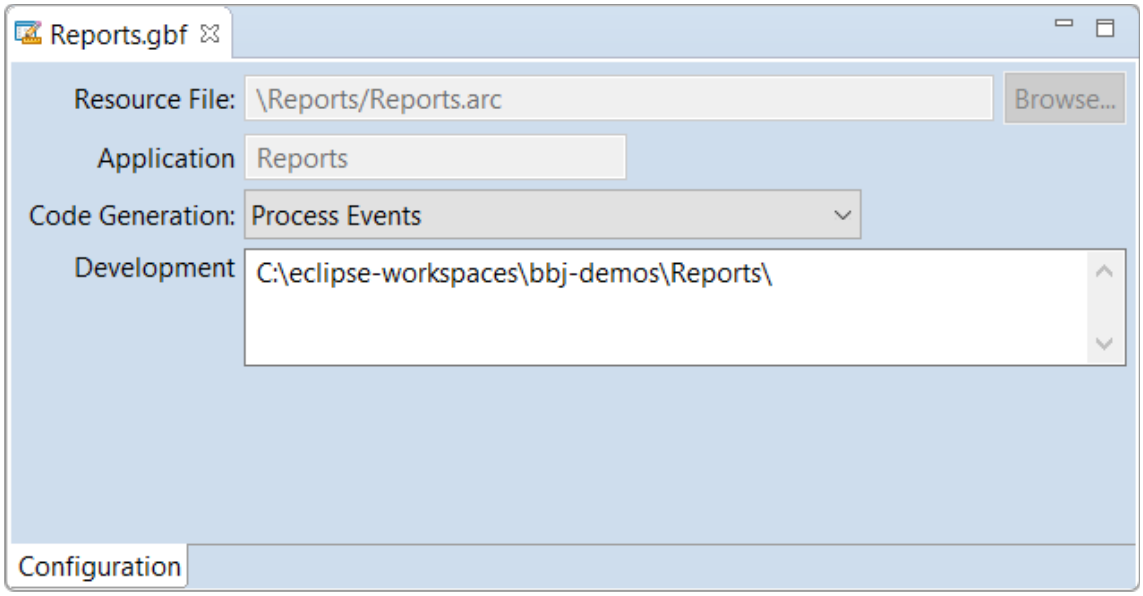

Figure 11. Appbuilder GBF Editor

BDT also opens two other views - the BDT ARC Inspector and the BDT Explored Objects. In the BDT ARC Inspector view, scroll down to the “107 (ChildWindowPtr):->101” entry and expand it. Select the “101.107->104 RefreshButton - BUTTON” entry under it following **Figure 12**.

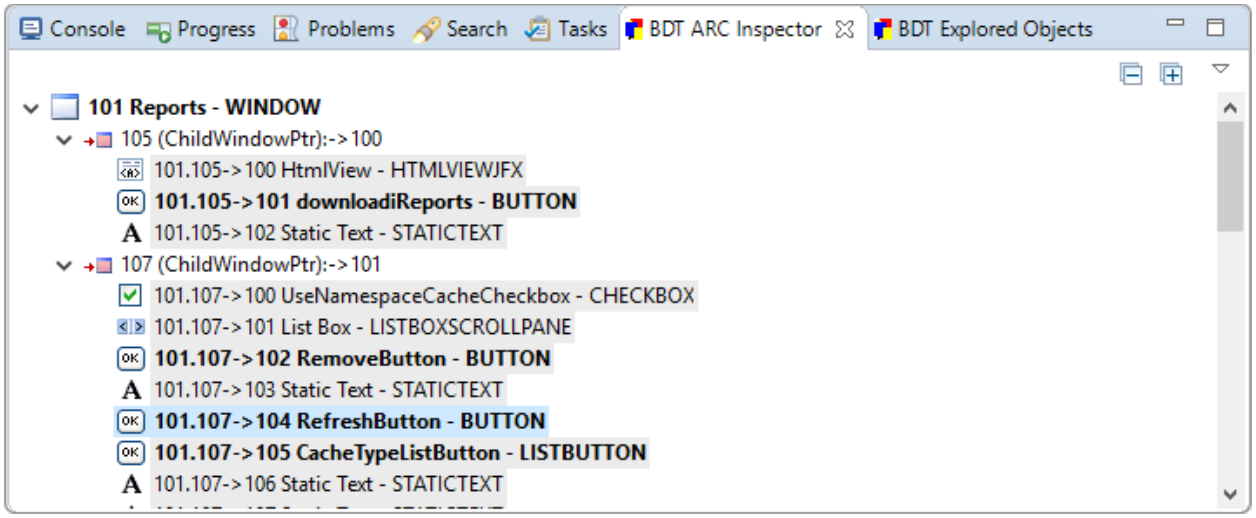

Figure 12. Selecting the RefreshButton in the BDT ARC Inspector

Then in the BDT Explored Objects view, click on the PUSH_BUTTON entry in the Registered Events list. **Figure 13** shows that as soon as you click on a registered event, BDT opens a new tab in the **`Reports.gbf`** editor window titled after the control and event you have selected.

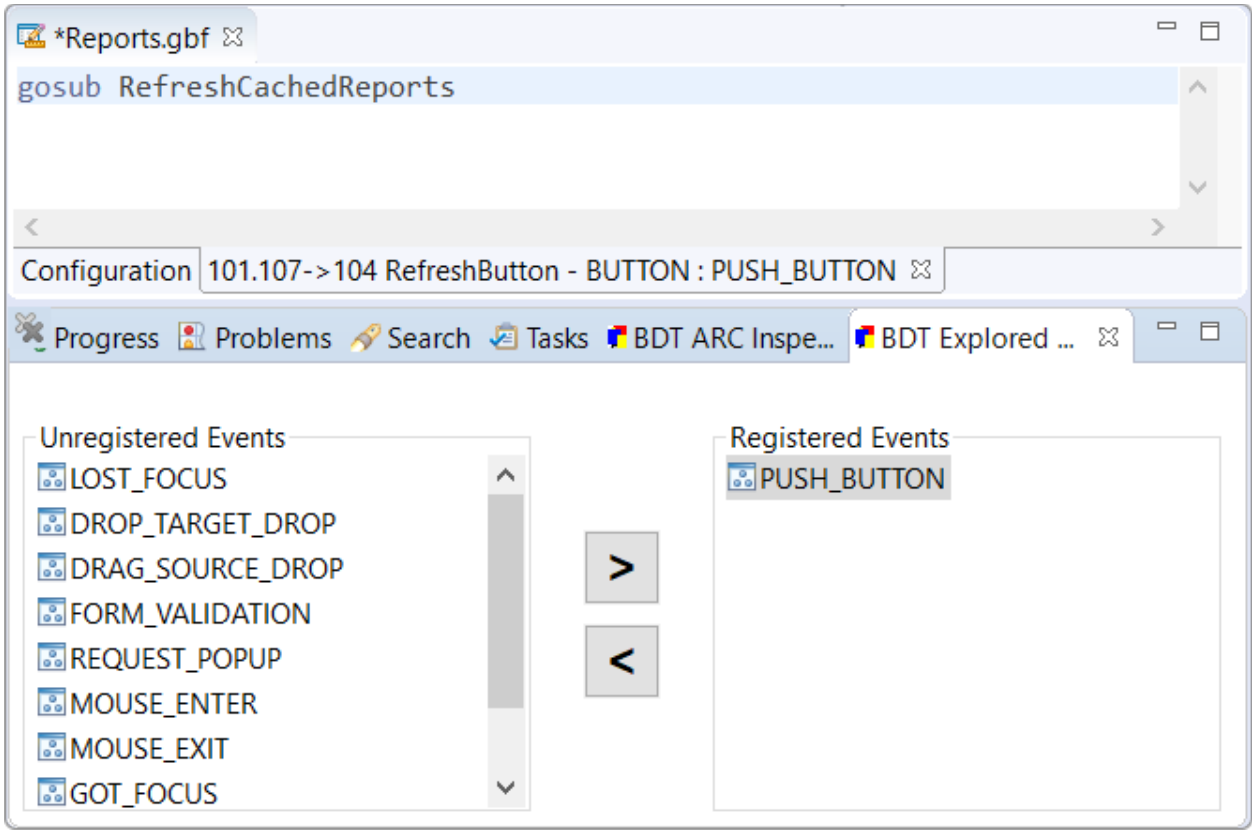

Figure 13. The Code for the PUSH_BUTTON Event

In this case, the event handler code consists of one line:

```
gosub RefreshCachedReports
```

In the **`Reports.gbf`** editor's tab for the PUSH_BUTTON event, add the following line of code:

```
x = MsgBox("All cached reports refreshed!")
```

to match **Figure 14**.

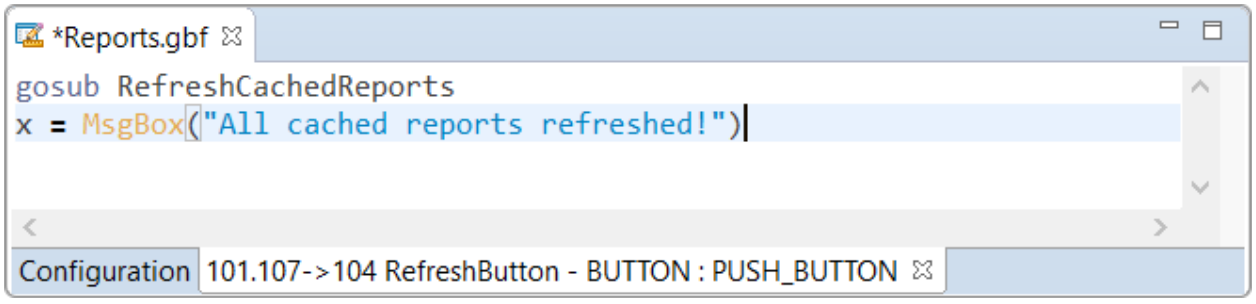

Figure 14. Editing the Code for the PUSH_BUTTON Event

Save this change, and click the [Build the active .gbf file] toolbar button (see **Figure 15**) to build **`Reports.gbf`** and generate new code.

Figure 15. The [Build the active .gbf file] Toolbar Button

Once the progress window disappears, building is complete. By default, AppBuilder builds .bbx output files that are tokenized, so you will now find a new **`Reports.bbx`** file in your project folder. **Note:** Eclipse does not always report "new files" to plug-ins when they appear, so you may need to manually refresh the Reports BBj Project. To do so in the BDT Explorer, right-click the Reports project entry and select Refresh (or hit the [F5] key). You will then see an updated listing as shown in **Figure 16**.

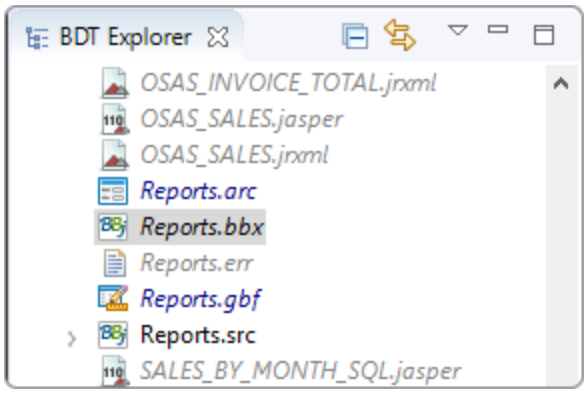

Figure 16. The Updated Reports Project's File List in the BDT Explorer

The **`Reports.err`** file in this case should be empty (no errors were encountered building the GBF), so you can either delete it or ignore it. To see our new event code executed, right-click **`Reports.bbx`** and select **Run As > BBj Program**. Once the BBj Reports display appears, select the Namespace Cache tab. Click the [Refresh] button, and our new message box appears similar to **Figure 17** below.

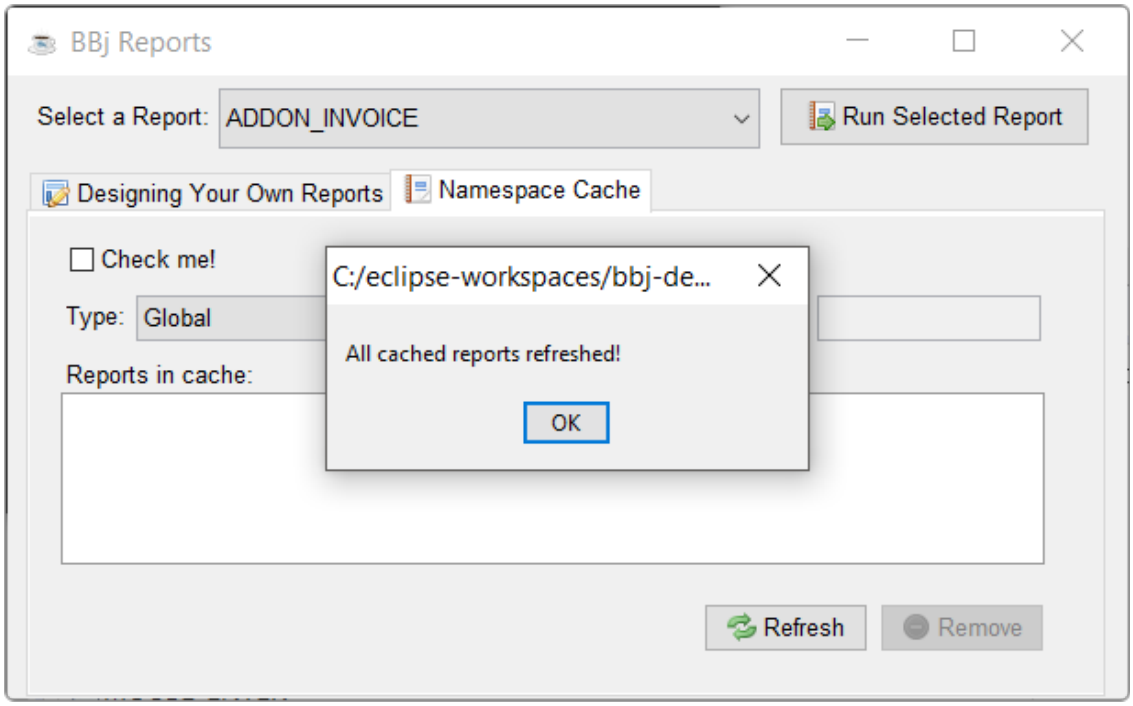

Figure 17. The PUSH_BUTTON Event's Message Box

Dismiss the BBj Reports application, and either revert your change to the PUSH_BUTTON event code or leave it, as you like.

Where Do You Go From Here?

As you have seen, there are many aspects of the BDT IDE that provide significant productivity enhancements - from making it quicker and easier to write correct code, to finding and fixing bugs, to creating tokenized output files for deployment. We have presented only a few of the ways you can use the BDT IDE to improve your BBj development. There are too many details to cover them all in one article, but for more information check out the much more detailed tutorial, Transitioning from NetBeans to the Eclipse IDE Guide. There you will find tips, tricks, and images, together with a Frequently Asked Questions (FAQs) section.

If, after examining the tutorial, you find that you have more questions, or that you are interested in applying what you have learned here to your own programs, here are some additional resources you may find helpful in your efforts to transition to BDT:

- For helpful discussions related to either the NetBeans or BDT IDEs, use the ide-user-group Google group. BASIS engineers also participate in these discussions.
- The IDE User Group wiki site is available as a developer community resource for sharing documents, links, source code, examples, images, or other artifacts that are useful with either IDE. Occasionally BASIS also adds links or other information to the wiki site that are helpful.

To learn about accessing and using these resources, see BASIS' Discussion Forums page.

As you can see, it isn't difficult to move your existing BBj program files from the NetBeans IDE to the BDT IDE. And once your programs are in BDT, maintaining them is easy and efficient. Give the BDT IDE a try and see for yourself!