Using NetBeans To Develop And Debug

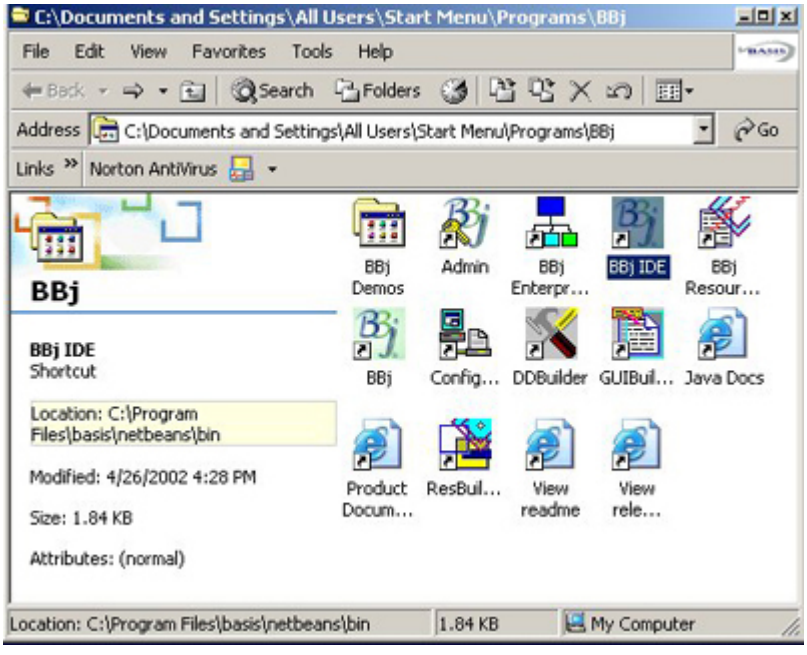
By John Schroeder

The NetBeans-based BBjIDE, new with BBj® Rev 2.0, is an all-in-one package for developing, testing, debugging and managing programming projects. In this first of a series tutorials on the BBjIDE, we will deal with developing and testing programs. We will also illustrate creating a program in the IDE editor, running the program, and discovering a bug.

Writing The Program

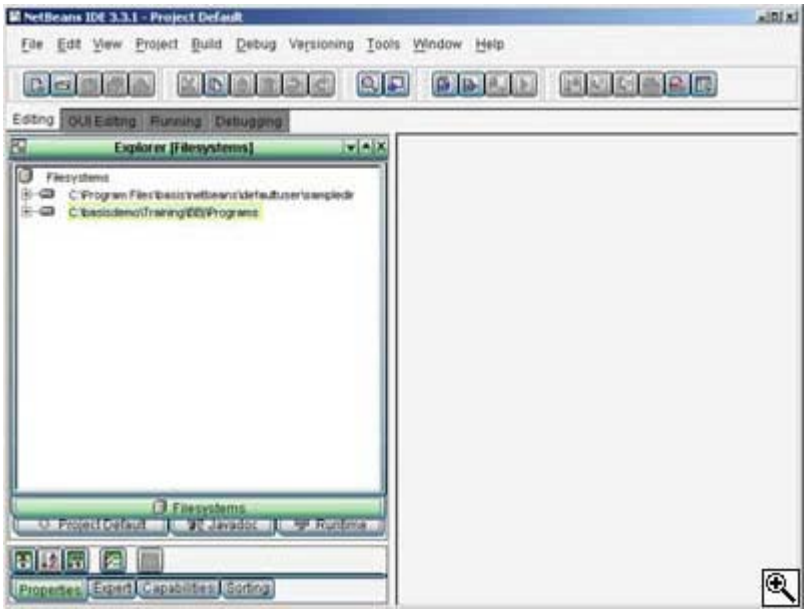
Let's start with the program requirements. We will write a program that displays the name, company name, phone number, and current balance fields from the "Chile Company" Customer file in a grid. When a user clicks on a row in the grid, the program will first determine which row has been selected and then display a message showing the name of the selected Customer. We will write the program using the BBj Object methods for creating and managing windows and controls. The main window and the grid control will be represented as objects, and the BBjVector object will be used to fill the grid with the necessary data.

Step one: Invoke the BBjIDE.



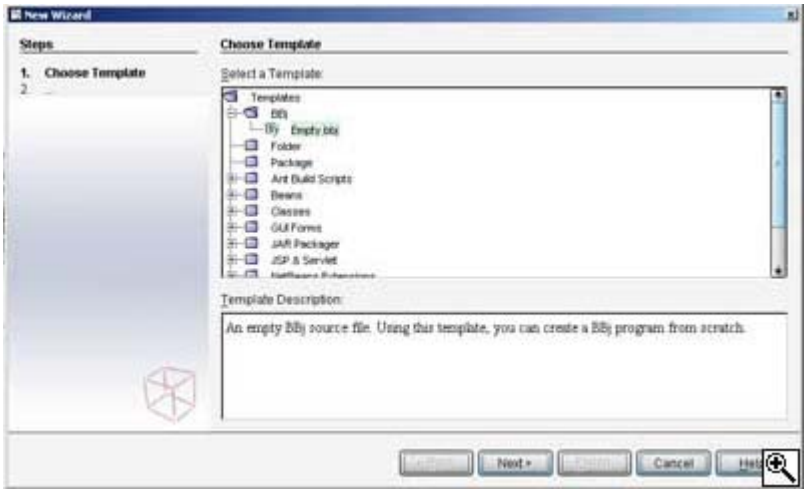
First we invoke the BBjIDE. This item in the program group was created when you installed BBj. It brings up the IDE with the Explorer window displayed, and the last directory accessed highlighted in the tree.

The IDE with the Explorer window displayed. The last directory accessed is highlighted.



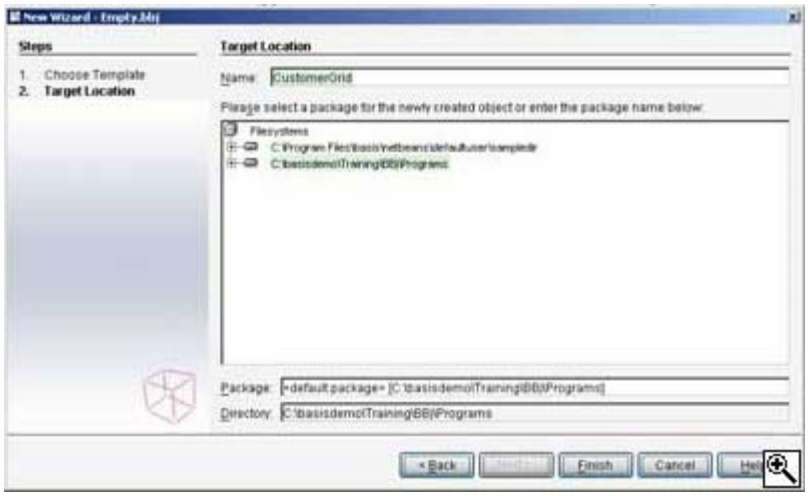
Create a new program in the IDE by clicking on the File/New menu or on the New tool button. This brings up the IDE New Wizard with a tree showing the types of files available in the IDE. Clicking on the "+" sign next to the BBj node expands the tree and displays the "Empty.bbj" node. Select this node for our new BBj program and click on the Next button.

After selecting the "Empty.bbj" note, click on the Next button.



The New Wizard "Target Location" screen displays a tree containing all the mounted file systems. The last directory that was used is highlighted. In this case, it is the Programs directory in our BBj Training directory. If the directory highlighted is not where you want to create the program, use the tree to select the directory you want. Then enter the name "CustomerGrid" in the Name field, and press the Finish button.

The New Wizard "Target Location" screen. The last directory that was used is highlighted.

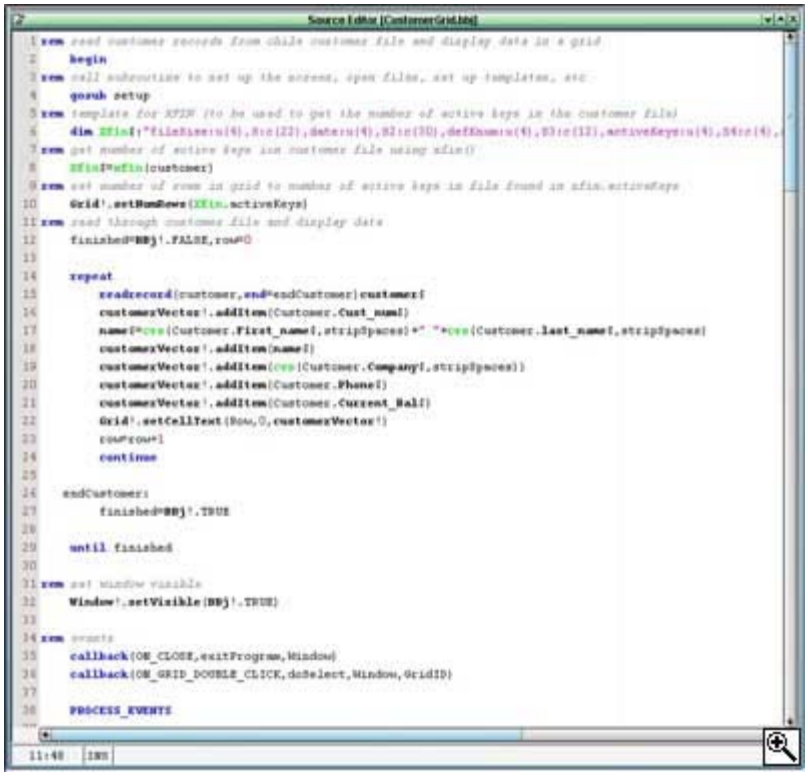


(You may access the full program in the "Cool Stuff" section of the BASIS web site. A portion of this program is listed in the IDE Editor, below.)

As you can see, the IDE editor colors the various parts of a BBj statement. The colors can be changed in the Tools menu under "Options/BBj Development/BBjModule Options/BBj Editor Settings". With syntax highlighting, you can see right away if you mistyped anything in your BBj statements.

This program is organized so that all of the screen building takes place in the "setupScreen" routine. Since this is not the main topic of our discussion in this article, we will concentrate on the section of code that reads the Customer file and copies the Customer data into the grid control. This portion of the program is shown below:

Here the code has read the Customer file and moved the Customer data into the grid control.



Building The CustomerVector!

After the screen is set up, the program opens the Customer file and uses the XFIN() function to return the number of active keys in the file. The XFIN() is an enhanced version of the FIN() function. It handles the new BBj XKEYED file type, as well as the traditional file types available in PRO/5®. The template for the data returned by the XFIN() is defined in line 6. In line 8, the XFIN() function is used to put the

Customer file FIN information into the variable, XFIN$. The template field holding the number of active keys in the file is used to set the number of rows in the grid in line 10.

The Boolean variable *finished* is set to FALSE, and the current row number set to 0, to initialize these values for the Customer file display routine.

This routine begins on line 14. In this routine, a Customer record is read from the Customer file, and the fields required in the grid are put into the CustomerVector! object. The CustomerVector! is a BBj vector object that was created to hold the data that is to be transferred into the grid. The Grid!.setCellText() method could be used for each entry in the grid, but each of these method calls involves screen I/O. Since moving the data into the vector involves no I/O, the process of using a vector is much quicker. The total time required to move the data from the Customer record into a vector, and then move the entire row into the grid, is less than the time required to fill each cell individually. Additionally, the grid is repainted only once per row instead of each time we change a cell's contents.

Event Handling

After the CustomerVector! is built for each row, the Grid!.setCellText() method is called to copy the Customer data into the grid at the row specified. Then the row is incremented, and the 'continue' verb sends control to the 'wend' statement. Since the end condition of the loop, the Boolean variable *finished*, is still FALSE, the loop continues. When an end of a file is encountered, the end branch sets *finished* to TRUE and the loop exits.

The event handling is done in the two callback statements. The first defines the event handler for the Close Box, the second for the grid double-click event. The 'PROCESS_EVENTS' statement then starts the event processing. Any events not set up in a callback are ignored. A close box event causes a gosub to the routine, starting at the statement labeled "exitProgram". A grid double click event causes a gosub to the routine "doSelect".

Having written the program, let's save it and run it to see if it acts as specified. To save the program, right click in the editor area and select Save. Click on the Save tool button, use the File/Save menu, or the keyboard shortcut, [CTRL]+[S].

Now it's time to execute our masterpiece. Once the program is saved, it can be run by pressing F6, or by right-clicking on the program name in the Explorer window and then selecting Execute. As you can see, the program creates the screen and grid properly and fills the grid with Customer file information.

The grid displays the same Customer in each row.

Unfortunately, the file displays the same Customer in each row of the grid. Now the process of debugging begins.. That is the topic for the next article in this series. Be sure to tune in next time for more adventures in "Using the New BBjIDE". Next time we will talk about using the Debugger.