

The Magical Reusable Dialog Wizard



The Merriam Webster dictionary defines dialogue or dialog as “a conversation between two or more persons; also: a similar exchange between a person and something else (as a computer).” It is in this second context that this article will delve further and describe a new BASIS tool that allows for the creation of the program that manages a dialog between a user and a computer.

The new **Dialog Wizard**, accessible from the IDE, is a modern and powerful alternative to AppBuilder. Its key advantage over AppBuilder is that it generates fully object-oriented BBJ programs from a resource file, automatically generating the callbacks and the method stubs. But, quite interestingly, it also preserves custom editing of the resultant BBJ code, allowing you to run a second or third pass of the code generator to leverage any changes you may have made to your resource file. It also optionally generates the automatic tool button integration to Barista. In addition, the Dialog Wizard gives you the ability to automatically test the resultant code, offering a launcher to launch a GUI or BUI version of your code. And, if that wasn't enough, it also generates Javadoc-like documentation from comment lines in the code!

Let's go through the steps and see just how quick it is to go from a resource file to a running program.

Resource File to a Running Dialog Program

After creating the resource file for a simple customer maintenance form, right-click on the resource name in the Filesystems navigation tree of the BASIS IDE and choose “Run Dialog Wizard” as shown in **Figure 1**.



By Ralph Lance
Software Engineer

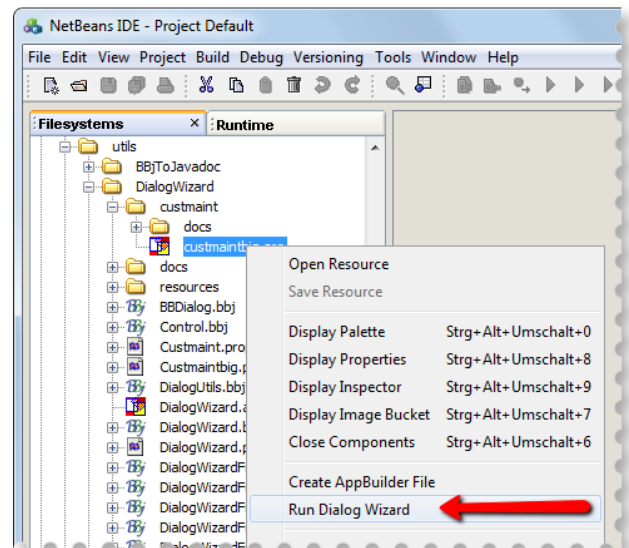


Figure 1. Filesystems option to Run Dialog Wizard

The Dialog Wizard launches and offers a suggestion of the program path to the program that is to be generated as shown in **Figure 2**; choose among the top level window IDs to steer this session of the wizard and then click [Next].

The Dialog Wizard's second frame shown in **Figure 3** lists the control names derived from the controls in the resource file and allows selection of the access scope of the member field, the variable name (or class member field) to be used in the program, the basic type of the variable, and a flag for whether the field is a required input. Click [Next].

This utility, as a minimum, requires a definition for the window close event, so select the control name that represents the top level window and either double-click on the “Close” event, or select the “Close” event and click on the right arrow button to add the Close event to the list of program events, as shown in **Figure 4**. Click [Next].

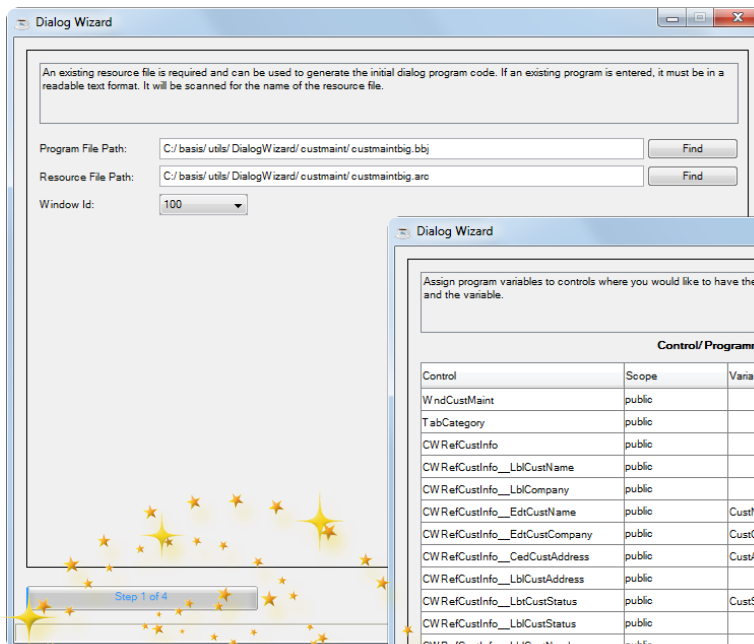


Figure 2. Dialog Wizard file paths

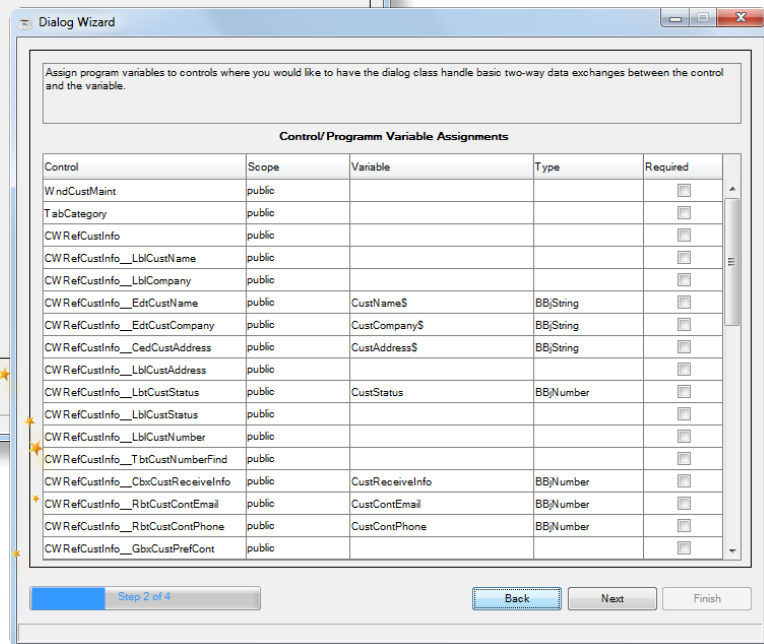


Figure 3. Control names from the controls in the resource file

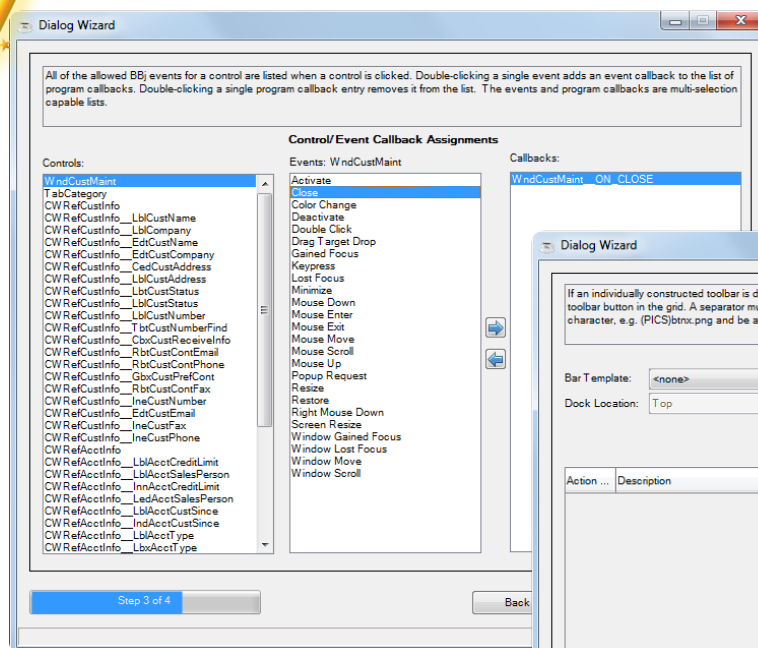


Figure 4. Add the Close event

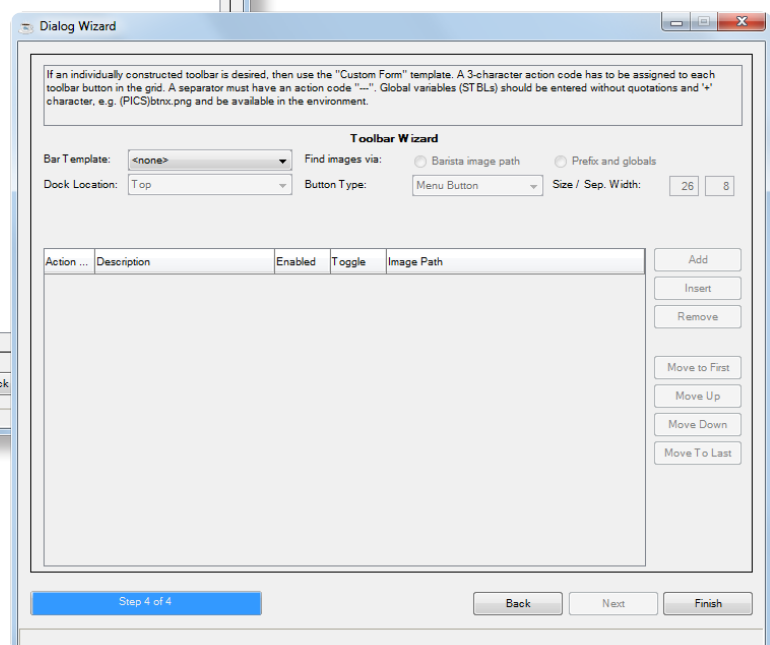


Figure 5. The final panel of the Dialog Wizard

The last panel (Figure 5) offers some selections that we'll visit a little later. For now, skip those by clicking [Finish] to generate the BBj custom dialog class program.

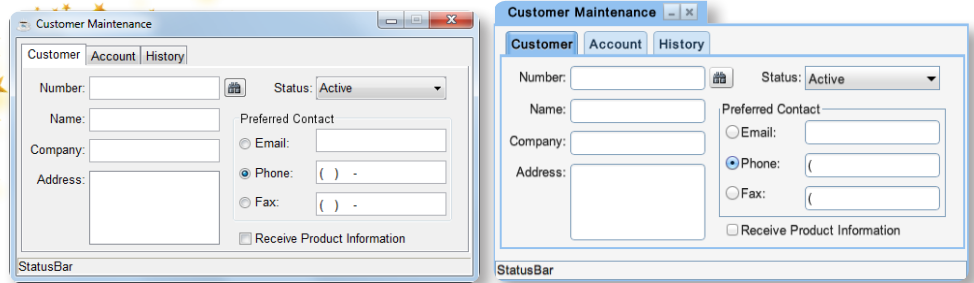


Figure 7. Test samples in GUI (left) and/or BUI (right)

Use a Dialog Wizard-generated Class in Conjunction With Barista

The Dialog Wizard is a productivity tool for individual dialogs that optionally can work with Barista. In other words, Barista isn't required to use the the Dialog Wizard or to run dialogs generated by it but it allows for the seamless integration of the dialog programs into Barista when so desired.

To illustrate this process, we'll run the Dialog Wizard again and make some changes in the toolbar section. We'll also add some code to an "Options" toolbutton to mimic an application enabling or disabling various toolbuttons to fit the context of the application.

First, we select where we would like to place the toolbar. In our example, we'll place it at the top of a maintenance form (see arrow #1 in Figure 8). In order to show the interplay between the toolbar in our dialog and the Barista MDI, we change the type of button used in the toolbar from menu to toolbuttons (arrow #2 in Figure 8). With this change, we can take advantage of the toggle function (arrow #3 in Figure 8) for showing interaction between our dialog class and Barista.

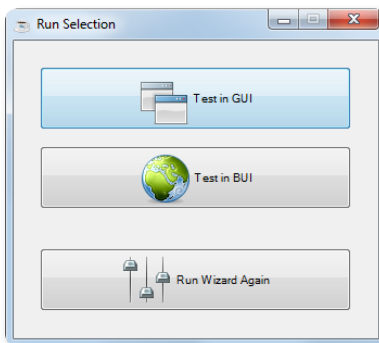


Figure 6. Test choices

One thing to note is the Dialog Wizard does not use data files, per se, for settings entered or chosen in the wizard for program generation. The program code itself contains this data. Through the use of identifiable begin and end code-blocks, for example, `rem /** DLGWIZ_DDX_BEGIN **/` and its matching end-block `rem /** DLGWIZ_DDX_END **/`, the wizard knows where to look for this data and parses what it needs from the program code. Any code outside of these blocks is ignored and will remain intact during the regeneration process.

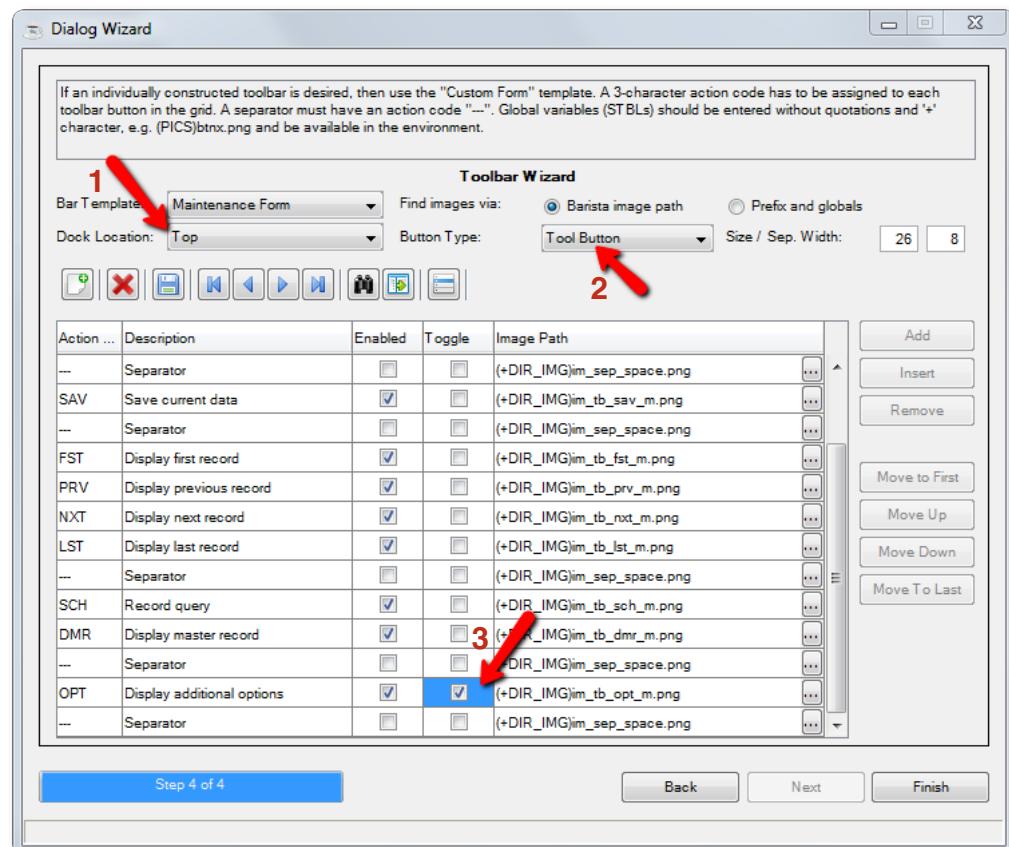


Figure 8. Defining the toolbar for Barista integration


```

rem /**
rem * Method onTBarBtnLSTPush:
rem * Event handler for the LST toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnLSTPush(BBjToolButtonPushEvent pEvent!)
methodend

rem /**
rem * Method onTBarBtnSCHPush:
rem * Event handler for the SCH toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnSCHPush(BBjToolButtonPushEvent pEvent!)
methodend

rem /**
rem * Method onTBarBtnDMRPush:
rem * Event handler for the DMR toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnDMRPush(BBjToolButtonPushEvent pEvent!)
methodend

rem /**
rem * Method onTBarBtnOPTPush:
rem * Event handler for the OPT toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnOPTPush(BBjToolButtonPushEvent pEvent!)
methodend

classend

```

Figure 9. Generated code for the toolbuttons


```

rem /**
rem * Method onTBarBtnOPTPush:
rem * Event handler for the OPT toolbar button
rem * @param BBjToolButtonPushEvent Tool Button Push
rem */
method public void onTBarBtnOPTPush(BBjToolButtonPushEvent pEvent!)
    actionList$="FSTPRVNXTLST"
    if pEvent!.isSelected() then
        #disableToolbarButtons(actionList$)
    else
        #enableToolbarButtons(actionList$)
    endif
methodend

```

Figure 10. Code to enable/disable the toolbar buttons

Click [Finish] to regenerate the program automatically. The code shown in **Figure 9** now contains method stubs for all of the toolbuttons on the toolbar.

Next, we add a little bit of code shown in **Figure 10** to the "Options"  toolbutton to enable and disable the four navigation buttons when the "Options" button is clicked.

These two methods are available for enabling and disabling toolbar buttons in the dialog as well as the corresponding toolbar button and menu item, for running the dialog class in Barista.

The new code lives outside any Dialog Wizard begin and end blocks, and is therefore preserved between program generations. If we now run our program in GUI, and click on the [Options] button, we will see the record navigation buttons toggle between an enabled and disabled state. Additionally, if we run this class inside of Barista (see **Figure 11**), we also see that the corresponding toolbuttons and menu

items in the Barista MDI match the enabled state of the navigation buttons on our form.

Summary

The new Dialog Wizard is great for rapid application development of your GUI apps, your BUI apps, and your custom dialogs that may not fit well into the core Barista data dictionary driven paradigm. You can create hand-crafted code and very easily plumb it into Barista. The Dialog Wizard encourages good development practices, as it results in documented object oriented code. It is also flexible enough to handle multiple executions against resource files that change over time as the program matures. See for yourself how helpful it can be on your next coding project! ■



See Dialog Wizard in the online documentation at links.basis.com/dialogwizard

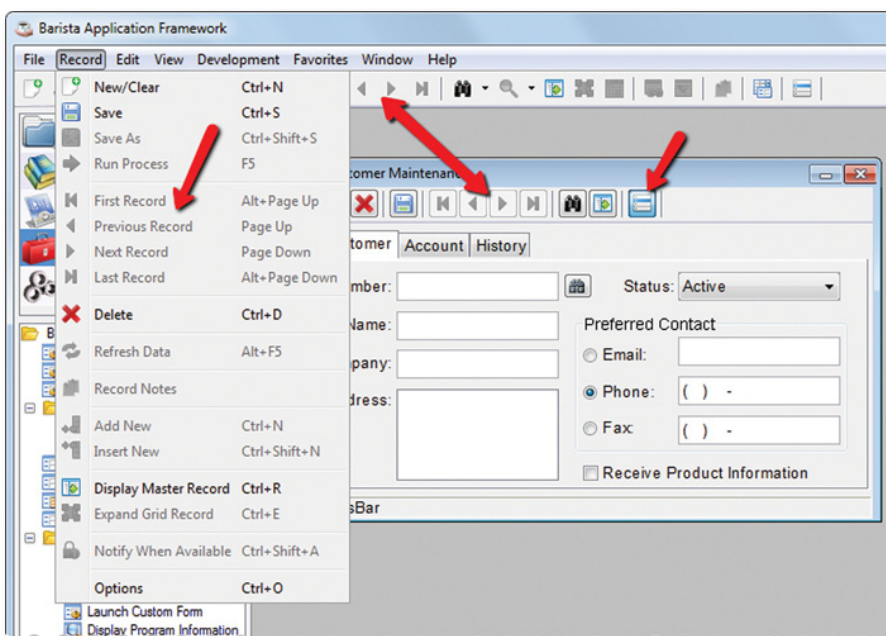


Figure 11. All record navigation buttons are disabled as a result of pressing [Options]