# Preserving Your Customizations


Like snowflakes, no two businesses are exactly the same and neither are their information processing requirements. How then, does one justify selecting a standardized off-the-shelf package to meet businesses' increasingly diverse IT requirements? One argument is that custom software is often more expensive throughout its entire life-cycle than a packaged solution, but packaged solutions are almost always inadequate in some areas. Up-front savings in a packaged solution can quickly dissolve as end users or IT staff have to devise external processes to make up for the package's shortcomings. On the other hand, customizing a packaged solution can leave users "coded into a corner," rendering the package ineligible for future upgrades and enhancements without losing the customizations.

If you are nodding your head in agreement and want to have the best of both worlds, read on. The Barista® Application Framework provides an answer to the dilemma! With Barista, you can customize applications in a way that preserves modifications through upgrades of the base product. Barista keeps track of customizations by saving them in a special project file structure outside of the base product's install location and then re-incorporates them after a product upgrade.

This article goes step by step through the customization and re-installation process using AddonSoftware® as an example.

## Step 1: Creating an Application

Before making modifications to a Barista application, you need to set up a directory structure in which Barista will save those modifications. Barista's Create Application wizard, as shown in **Figure 1**, collects information about your project such as the top level directory, your company ID, a description for the project/application, etc. and creates an application-area for the modifications.

Barista uses the information you provide to create default STBL values for the barista.cfg and BASIS config.ini files **(Figure 2**):

When the wizard has collected all necessary information, Barista creates a file structure for your project and prompts you to run the Auto-Synchronization process (**Figure 3**). This process forwards information provided in the wizard into your barista.cfg and BASIS config.ini files (**Figures 4a and 4b**), and you're ready to begin making customizations. >>

***By Chris Hawkins***
*Software Engineer*

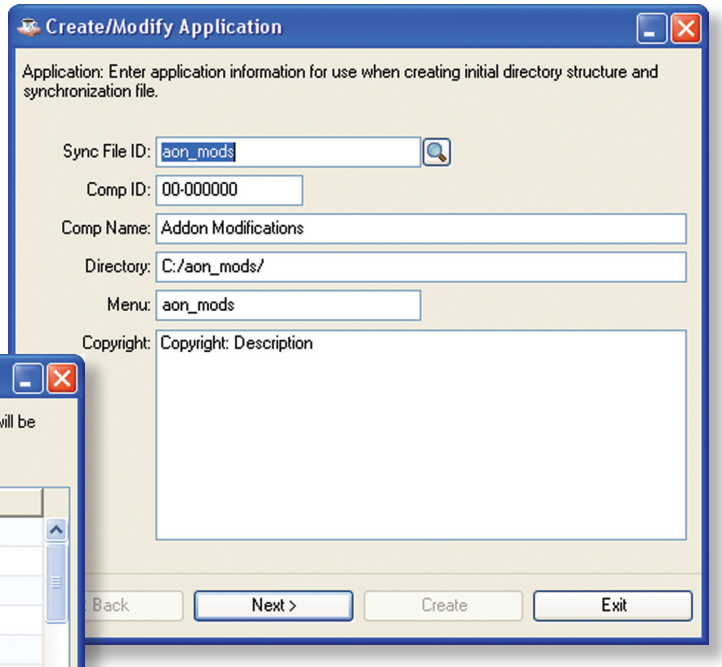


Figure 1. Create Application wizard collects information for the project

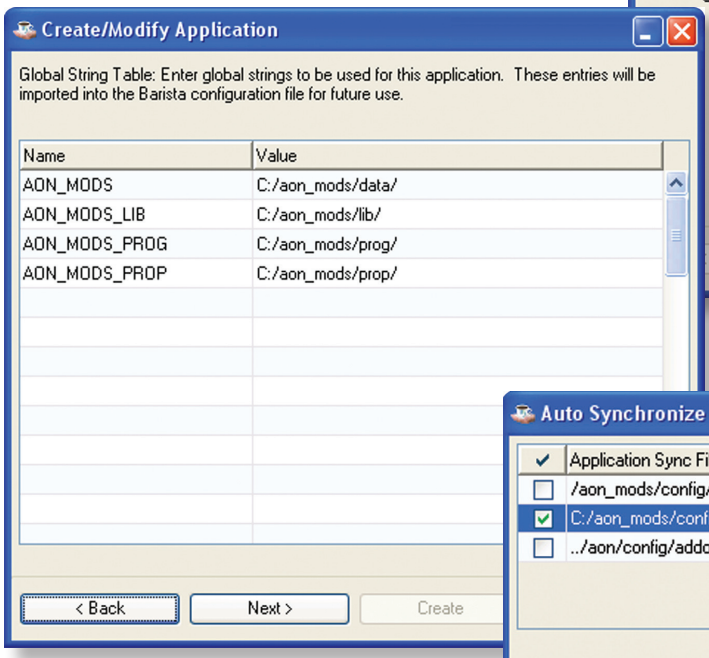


Figure 2. Barista provides default configuration settings

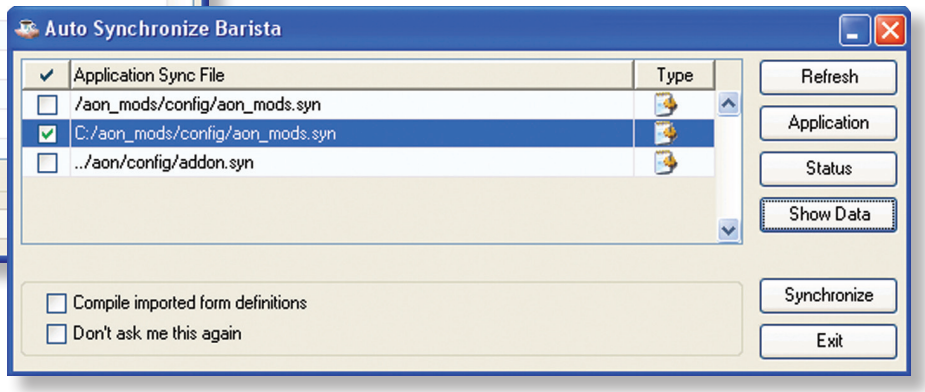


Figure 3. Auto-Synchronize new project into Barista

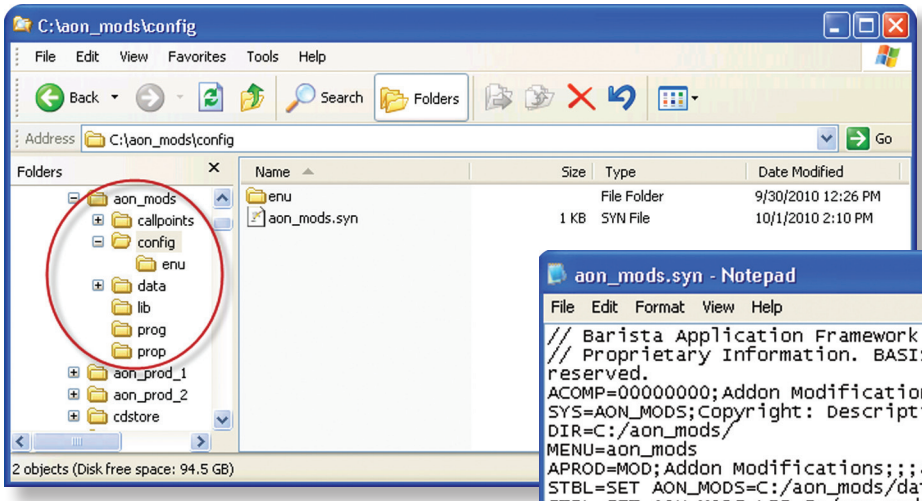


Figure 4a. Barista creates directory structure for new project

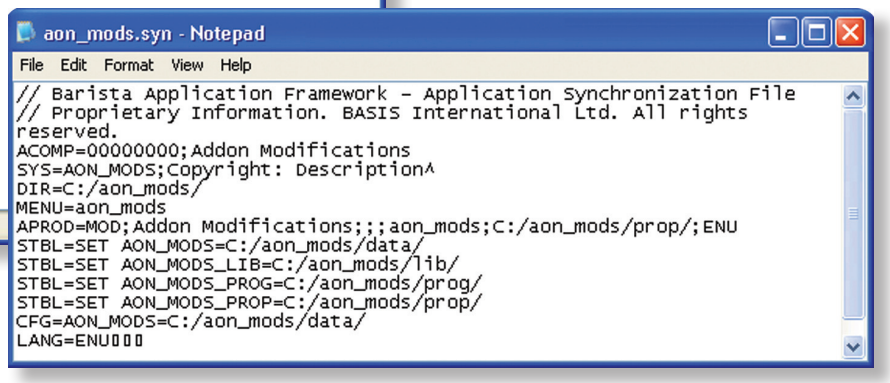


Figure 4b. Information supplied during wizard is captured in .syn file for new project

## Step 2: Customize the User Interface

In order to customize forms or business-logic callpoint code, call up the Form Manager and right-click on your project name. The padlock icon that appears over the top of your project icon (see **Figure 5**) indicates that Barista is in "replication mode." Any form or callpoint changes you make to the base product will also be saved into your project area.

First, we'll create a brand new table and form directly in our project area to store new Vendor Category codes and descriptions (i.e., categorize what we purchase from this vendor: raw materials, inventory items, services, etc.). Then we'll alter the standard AddonSoftware Vendor Master form, adding a Vendor Category field that validates to our new table as shown in **Figure 6**. When we save and build the form, the Barista resource file (.xml) is saved in the standard product's **`/data/bar`** folder, as well as in your project's **`/data/bar`** folder.

## Step 3: Add Custom Business Logic

In order to make sure that we always have a value in the Vendor Category field, we can also add custom callpoint code. In our example, we set the Vendor Category to "UND" if no code is yet defined.

When in replication mode, you can see the callpoint code for the standard product, but can't modify it directly. Instead, you add code that executes before, after, or instead of the standard callpoints. Barista runs the Before callpoint code, then any code that's part of the standard product, then the After callpoint code (see **Figure 7**). >>

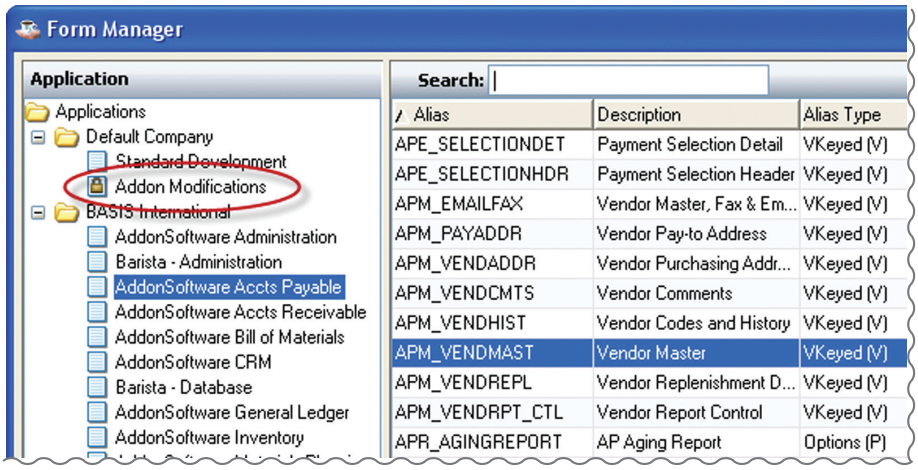


**Figure 5.** Development in replication mode

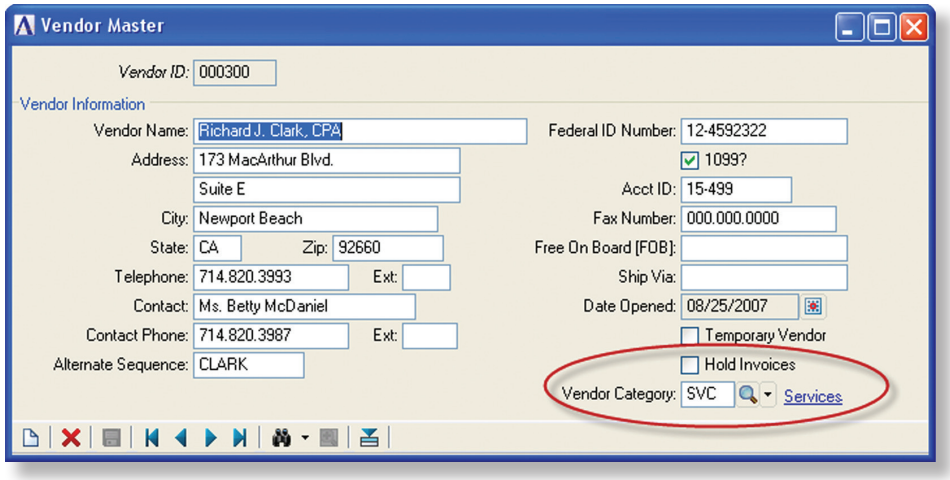


**Figure 6.** Revised Vendor Master form: moved Hold Invoices to the right and added new Vendor Category field

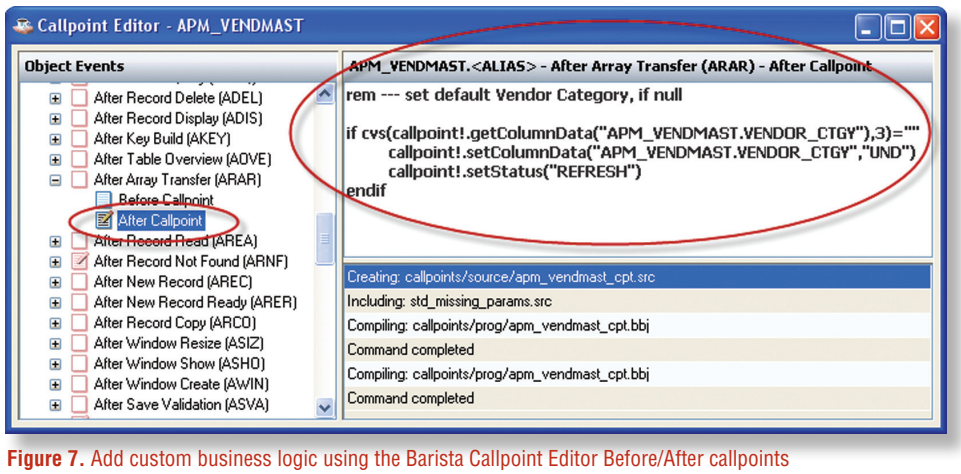


**Figure 7.** Add custom business logic using the Barista Callpoint Editor Before/After callpoints

To run your code only, that is, instead of the standard callpoint code, use the Before callpoint along with the method `callpoint!.setStatus("SKIP")`.

## Step 4: Create Custom Reports

In addition to modifying the user interface and business logic controlling the forms, you can also customize or write new "back-end" code such as reports, updates, or publics. Extending our AddonSoftware example, we'll create a modified version of the Vendor Name and Address listing that also shows our new Vendor Category. First, we need to alter the "Run Program" setting in the Option Entry form for the report so that Barista runs our customized report as shown in **Figure 8**. Remember, because we're in Replication mode, the Barista resource file for the Vendor Name and Address listing will be saved into our project area for safekeeping.

Next, we'll use a text editor such as the BASIS IDE to create a copy of the standard report in the "prog" directory of our project area (**Figure 8a**), and then add code to open and retrieve data from our Vendor Categories table.

Our custom report shown in **Figure 9** is now totally contained within our custom project area, so we'll be able to re-incorporate it after our next upgrade.

## Step 5: Re-install Customizations After an Upgrade

Barista's ability to facilitate this sort of customization adds value to the product and also ensures that the product isn't frozen in time! All of the modifications to >>

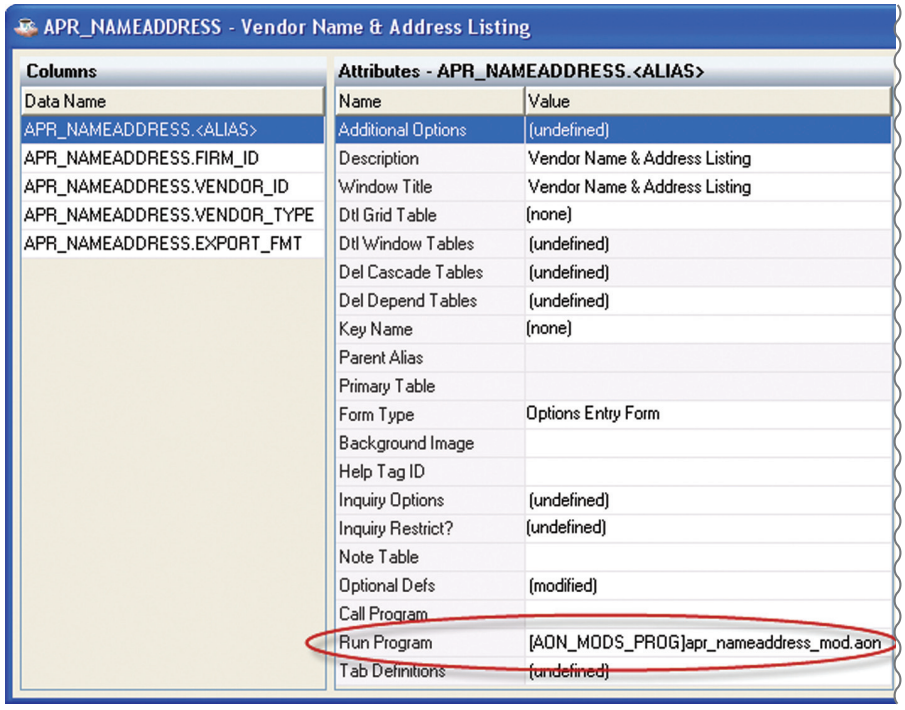


**Figure 8.** Changing the "Run Program" tells Barista to run our customized report

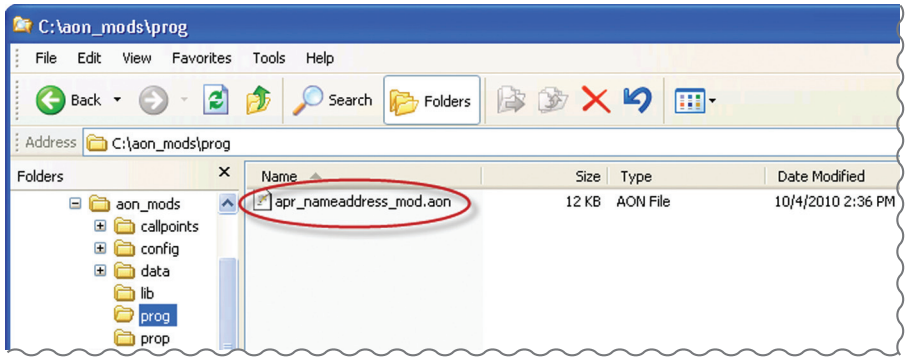


**Figure 8a.** Place back-end code in your project's "prog" directory


**Figure 9.** Customized Vendor Name and Address Listing

the base product are preserved in our project area so we can upgrade the base product and then use Barista's Install Application wizard to re-incorporate our customizations (**Figure 10**).

After an upgrade, simply point the Install Application wizard to our project area. Barista analyzes the resource files in the project (`<project>/data/bar/*.xml` files) and compares information in those files with the current dictionary to check for possible conflicts. If it finds issues or conflicts, you can print a report that lists them and decide if the issue level should be lowered (i.e., no longer considered a critical issue) or if you need to make changes in the project in order to re-synchronize it into Barista. The latter may occur if, for example, you added a new field as part of your customizations, and that new field has now also been added to the upgraded product.

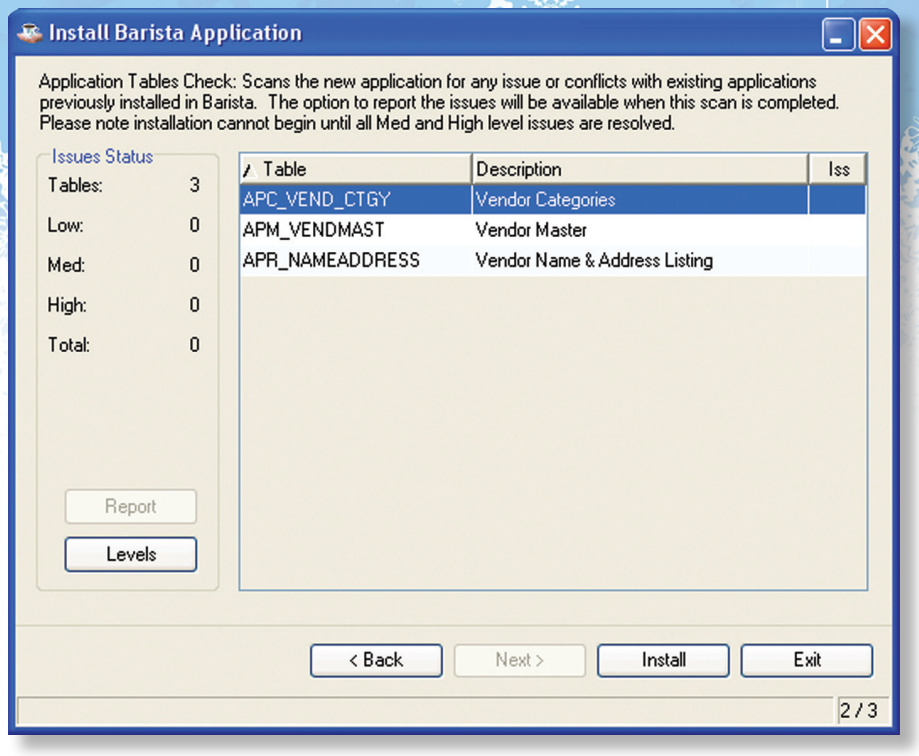


**Figure 10.** Barista Install Application Wizard

When no impeding issues remain, Barista processes the project's resource files, incorporating their data back into the current dictionary, then rebuilds any tables, forms and callpoints as necessary. When the wizard is complete, your modifications are once again part of the base product.

## Conclusion

The Barista Application Framework gives you the tools you need to provide customized solutions that can be preserved through the upgrade cycle. Now you can help the user address their unique business needs without the fear of becoming frozen in time. So.... let it snow! ■

*next article* >>>