Beans Binding and Swing Application Framework support, and other new features you’ve probably been dreaming about having in your IDE.
UI Design in NetBeans 6.0

opers gain more from the Swing Application Framework enhancements in NetBeans, but Platform developers should not feel left out. The latter already have a Swing-based application framework in the NetBeans Platform, with many more features than JSR 296 provides. All gain much with Beans Binding support, however.

Beans Binding support

Beans Binding allows you to escape the common monotony of writing code to copy data from user interface components to data classes or JavaBeans and vice versa. With the new Beans Binding enhancements, you can right click on a UI element and access the Bind context menu item. For instance, selecting this item for a javax.swing.JTextField will show the preferred bound property of text (see Figure 1). Other properties can be accessed through the

Figure 1

The Beans Binding menu item.

Figure 2

Setting Beans Binding options for a JTextField.

Beans Binding and the Swing Application Framework

One of the most talked about and innovative features since NetBeans 5.0 is Project Matisse or the Form Editor. Many would agree that Matisse is the best user interface designer across different IDE categories and technologies. A good number of developers will start or have started using NetBeans because of it.

With software, there is never-ending room for improvement and growth. In NetBeans 6.0, familiar and new developers will find many new Matisse features to help improve Swing application development, as you’ll see in this article.
property inspectors Binding tab (Figure 2).

Once you've chosen the property to bind, a dialog is displayed, where you can select the target JavaBean. You can then enter an expression using the Beans Binding Expression Language. The syntax is much like the JSP EL. Aside from entering the expression by hand, a nifty selector, accessed as a drop-down list or combo box, allows developers to quickly build the expression by drilling down through properties and sub-properties.

When the application is run, the selected UI component updates its bound component when focus is lost or the user presses Enter. The main point is that the developer no longer has to manage this and other operations with a load of event handling code.

Swing Application Framework support

The Swing Application Framework provides mechanisms to quickly build complete desktop applications. NetBeans 6 takes it further with integrated support for the framework in the IDE, while providing a set of standard icons such as Copy, Paste, and Cut. This is better than piece milling an application from Swing components, coming up every time with a separate solution for starting an application or shutting it down, basic actions or events, custom icons, a resource manager, session storage, etc.

To create a Swing Application Framework project in NetBeans 6, select File>New Project and, under the General category, choose the new Java Desktop Application project template. At the time of writing, there are two application templates available: Basic and Database (see Figure 3). Others will be included in future releases. For example, thought is being given to templates for creating web client applications based on the Swing Application Framework.

The Basic template generates a regular Swing application with simple features such as Cut, Copy, Paste, Save, and New. Simple applications like Notepad or KWrite can be easily created with this template (see Figure 4).

The Database template allows users to create CRUD database applications. Along with Apache Derby/JavaDB or HSQLDB, this can be like Microsoft Access on steroids.

To me, the Swing Application Framework support works best for developing simpler applications. This may get better in the future with JSR 277 (Java Module System). However, a module system alone does not provide a framework with all the components and utilities offered by the NetBeans Platform. For more complex applications, I advise you to build on the NetBeans Platform. A path from Swing Application Framework-based applications to Platform-based apps is being pondered, but nothing is concrete in this regard at the time of writing.

The Swing Application Framework and the NetBeans Platform are both Swing frameworks. The Swing Application Framework provides application lifecycle management, session management (e.g.
windows are stored in the same locations when the application restarts), a resource manager, actions, storage, an application context, and synchronous and asynchronous tasks. The NetBeans Platform provides all these plus numerous other features, including a powerful module/plug-in system which allows modules to install their own UI menus, actions, and services, among other application items.

More new features

Additional new features in NetBeans 6 include a new visual menu designer, protected code customizer enhancements, centering of components and improved Free Design preferred-gaps and copy-paste support.

Other features added to version 6 have been made available for NetBeans 5.5 developers as an update. These include automatic internationalization, visual localization, a preview with look and feel support, relative font definitions, and dragging of components from the projects explorer to the UI – as well as a context-sensitive help bar, and a pre/post declaration code editor. In the upcoming IDE release, all these features will be fully integrated, and come out of the box.

Visual Menu Designer

Previous versions of Matisse had limited application menu design support; you needed to use the Inspector window to create menu items and sub-menus. Now menu components can be selected and edited visually in the UI designer. This helps in a couple Java Desktop Application ways: it’s clearer which menu and menu item is being edited, and the form doesn’t have to be previewed or the application run to see what the menu will look like at runtime. See the Visual Menu Designer in action in Figure 5.

Protected Code Customizer Enhancements

NetBeans uses the concept of protected code. This is used by Matisse so that the generated code is not changed and possibly broken by the developer, allowing the visual designer to continue working. Sometimes, though, this is a little too strict, and restrictions on protected code are a common complaint in the NetBeans mailing lists.

In NetBeans 6 this will be much better, and you’ll be able to change the protected code sections in many ways. These help you get around issues of timing actions and method calls, with initialization and property/bean configuration, for example.

Here are some options for modifying protected code, all accessible through the Properties window Code tab.
After-All-Set Code – Included after the component is completely set-up.

The new Pre/Post Declaration Code Editor is more of a must-have than a development boost. As of Java 5, developers can now annotate different things in Java source code, and for libraries or technologies requiring annotations, Matisse must allow the developer to somehow set these annotations. Annotations can also be added through the Properties window’s Code tab, in the Pre-Declaration Code field; and there’s a Post-Declaration Code field available.

NetBeans 6 comes with a new code customizer, which lets you inject source code more easily into the protected sections. It’s accessible by right clicking on the UI form in the designer and

(also see Figure 6):

- Post-Listener-Code – Included after all properties of all beans are set and all listeners are added.
- Pre-Adding Code and Post-Adding Code – Included before and after the component is added to the parent container such as a JPanel or JFrame.
selecting Customize Code. A dialog with a Java editor pops up. See Figure 7 for an example.

Centering of components
Aligning components along a center axis just got easier. Matisse now allows a group of selected components to be centered down the axis of the first selected component. Horizontal and vertical centering are available. Currently, more than one component must be selected; then all are centered on the widest selection. Another option will allow centering components horizontally and vertically in their parent container.

More gaps
Before NetBeans 6, the Free Design layout manager supported a single preferred gap for component placement. In 6.0, three preferred gaps are supported. A preferred gap is the preferred spacing between components, and is available on all sides of a component for quick and elegant placement, as shown in Figure 8. For developers who may like more control over spacing between components, having three choices comes in handy.

Better copy/paste support
While using the Free Design layout,
copying and pasting produces a horrible result in versions before 6.0. It places all copied components, no matter the number selected, at location [0,0] on the UI form. This problem has now been solved. Instead of messing up the layout, copies are now pasted to a manageable offset of the original components. The components also keep their layout after copying (see Figure 9).

Automatic Internationalization and Visual Localization
Previous releases of NetBeans required a few extra steps to internationalize and localize a UI. Now the process is streamlined. In previous versions, you had to setup each individual UI element to pull values from the correct resource bundle. Then to actually localize the bundle you needed to create separate bundle files or localized entries manually, enter the text for the correct locale, and format the file accordingly (using a different encoding, for example).

In NetBeans 6.0, you can now ask Matisse to automatically internationalize the application during UI design. This means that for each resource that would normally be internationalized by hand Matisse automatically adds the value supplied in the UI designer to a resource bundle. This is done for all UI elements.

The visual localization feature works in harmony with automatic internationalization. You can right click the top form node in the tree of the Inspector window, locate the Design Locale combo box in the Properties window, and then select a locale or add a new one. Once the Design Locale is selected, you can simply edit the text

Figure 9
Improved copy/paste support in Free Design (components were pasted multiple times).

A JSR 296 Swing Application Framework implementation

Figure 10
Access the Automatic Resource Management combo box to set the resource management style to Internationalization, and Design Locale to set the locale used while editing the UI in Matisse. Change the locale and re-edit the UI to localize the user interface. Nice and easy!
in the UI as if normally editing a form. The localization for the Design Locale takes place automatically with the values entered into the UI. The locale can be changed and the form re-edited to set the values for the newly selected locale. It is as simple as it sounds to create a fully internationalized UI with NetBeans 6 (see Figure 10).

Preview with look and feel support

Before NetBeans 6.0, there was no mechanism to allow the project’s look and feel to be set in the designer for a regular Swing application. It was left to the developer to add the appropriate code to set the application look and feel.

Setting the look and feel of a regular Swing application with source code is quite simple. However, without the Form Editor allowing the design look and feel to be set, the developer has no way to see how the application will look at runtime with different look and feels. This will no longer be a problem.

In NetBeans 6.0, forms can be previewed with any available look and feel. In the Inspector window, the developer right clicks on the form containers node, or on the node on the same level as the Other Components node and just below it (its sibling), which has the name of the forms extended class such as JFrame, JDialog, or JPanel. Then the Preview Design sub-menu is rolled over and the look and feel is selected. The form is then previewed with the chosen look and feel.

A right-click on the form in the UI designer also makes the Preview Design sub-menu accessible.

Relative font definitions

Have you ever wanted to define your UI fonts just a little larger or smaller than the overall system or application font? This would come in handy in that fonts would not have to be set for each component. Instead, fonts for specific components could be set to a different relative size, or have a different style than the main application or system font.

In the latest version of Matisse this is quick and simple. You select a UI component on a form, and in the Properties window click the button to the right of the font property. The font editor appears. There, you can select the checkbox “Derive the font from the default font” and choose the font style. The font can be sized relative to the default font; the size can also be made absolute. Regardless of the size, the font is not hard-coded and will be determined
at runtime. This feature uses the `Font.deriveFont()` method.

**Dragging components from Project Explorer to the UI**

In previous versions of NetBeans, the developer had to install UI components from developing projects into the Form Editor’s Palette to use them in UI forms. This created different issues with modifications to the components, and complicated using components from sub-projects in project UIs. Things have gotten better, however.

Now you can work on a library as a sub-project or dependency. You create a new form in the parent project, and the sub-project UI controls/classes can then be dragged from the project hierarchy and dropped onto a form without adding them to the palette or the global IDE. This is essentially project-level palette items: a nicer and cleaner solution.

**Context-sensitive Help Bar**

Who says you can’t teach an old dog new tricks? The new context-sensitive help bar (see Figure 11) can help new and familiar developers alike. New users will probably find it more useful, as it can quickly bring them up to speed with the Matisse features. The new help bar provides contextual hints about what can be done with the selected component or current feature being used. Even seasoned users may find new shortcuts or features they did not know existed.

**Conclusions**

Matisse is part of the reason for the NetBeans community’s recent growth spurt, and it gets better and better with each release. The 6.0 release is no different, and the latest features are a great boost to Swing application development efficiency. Everyone should try it.

NetBeans 6.0 Matisse feature development is ongoing, and other features are currently in the works. Most notably, the possibility to fix refactorings for generated UI code should be ready in the final version, along with many other new features and improvements. 😊