Java GUI Builders

By Mitch Stuart

Document Version: 1.4, 06-Apr-2005

Contents

1 Introduction

This article discusses and compares tools for building Graphical User Interfaces (GUIs) in Java. It describes my experiences and results in trying to find tools for rapid development of Java GUIs.

1.1 Discussion Thread

There is a JavaLobby discussion thread for this article, where you can view and post comments and questions.

1.2 Evaluation Scope and Methodology

Using web searches, I identified more than 20 Java GUI builders to investigate. I read each product web site to determine the feature set, licensing, and pricing. I did other web searches to find existing reviews or opinions of the products, and also to

discern vendor/community robustness. In this step, I eliminated some of the products from further consideration (see the Product Summary for information on why some of the products were not selected).

With the remaining set of finalist products, I downloaded and installed each one. Then I tested them by creating panels with various sets of components.

1.3 Evaluation Criteria

I did not use a formal evaluation matrix or checklist. However, the following items were considered when evaluating each tool:

- Quality of resulting GUI. Does it look professional and organized? For example, do all the fields line up properly? Does the form behave properly when resized (for example, expanding any growable fields as appropriate).
- Development productivity for application development. How quickly can you go from a "blank slate" to a working application?
- Development productivity for maintenance. For example, let's say you have created an application with a form. Now you need to modify the form to add a couple of new components (labels, text boxes, etc.). Is the mechanism for maintenance / editing of the form consistent with the original development? Or do you have to use different techniques when modifying existing forms?
- Cost and other business considerations. Is the product commercial, free, open source? Is it being actively developed / maintained? Is the vendor and/or community robust?

2 GUI Builder Background

Before jumping into the product discussion, in this section I will present some technical background that will be useful in understanding the similarities and differences between the GUI builder products.

2.1 GUI Components: Swing vs. SWT

Swing is the "built-in" GUI component technology of the Java platform. Swing is the successor to the AWT technology that was provided with the early releases of the Java platform. In one sense, Swing replaces AWT. For example, in a Swing program you would use javax.swing.JTextField instead of java.awt.TextField. In another sense, Swing builds on AWT: JTextField is a descendant of java.awt.Container, and many non-component AWT classes (such as layout managers) are used in Swing programming.

SWT (the Standard Widget Toolkit) is an alternative (competing) GUI component technology that is part of the Eclipse project. There is a large and growing community that advocates SWT over Swing for Java GUI programming.

In my own programming, I prefer using Swing, so this article focuses on tools that work with Swing. If a tool happens to support SWT also, I will mention that fact (if I'm aware of it), but I have not tested the SWT capabilities of any of these tools.

2.2 Layout Managers

One of the key decisions in determining how your forms will look, and how easy it will be to create and maintain them, is what layout manager you use. It is quite common to have multiple panels and nested panels within a single form and certainly within a full application. So you may use several different layout managers. But for your "mainstream" forms with labels, data entry fields, etc. that you need to layout and line up, you will likely use a complex layout manager that allows you to layout many components cleanly in relation to each other.

Here is an overview of some common layout managers:

- **Simple layouts**. BorderLayout, BoxLayout, CardLayout, FlowLayout, and other similar layouts are simple layout managers that are suitable for laying out a small number of components or nested panels. These are very simple to use and tend to have only a small number of properties to set. For example, for FlowLayout you can set the alignment, horizontal gap, and vertical gap.
- **GridBagLayout**. A complex layout manager that is part of the standard Java distribution. If you are building a complex Java GUI, and not using a third-party layout manager, you will tend to end up using this layout. It is very flexible but difficult to work with (there is an amusing animation called Totally GridBag illustrating this). One of the key criteria for evaluating a GUI builder is determining how much it simplifies the process of working with GridBagLayout.
- **SpringLayout**. A simple but powerful layout manager that is part of the standard Java distribution (as of Java 1.4.x). Sun's Java tutorial says: "The SpringLayout class was added in v 1.4 to support layout in GUI builders." However, of the GUI builders that I looked at, only Swing Designer currently supports SpringLayout, as far as I can tell.

  In an interview, NetBeans team member Tim Boudreau seemed to indicate that there would be SpringLayout support in NetBeans 4.1. However, in a chat session, NetBeans team member Larry Baron stated "The Form editor from 4.0 will migrate unchanged to 4.1. We are looking at improvements in releases past NB 4.1".

  So although SpringLayout looks intriguing, it does not yet seem to have sufficient support among GUI builders to have a major impact on the evaluation here.
- **FormLayout**. A third-party open source layout manager from JGoodies. FormLayout has capabilities similar to GridBagLayout, but is much simpler to use. The JGoodies Forms package includes not only the FormLayout layout manager itself, but also a set of "builders" (non-visual helper classes) to assist in building panels. FormLayout seems to have the broadest support among GUI builders of any of the third-party layout managers.

- **Other layouts**. There are other layout managers capable of laying out complex forms, like TableLayout, ExplicitLayout, RelativeLayout, EasyLayout, and SGLayout. However, these do not have broad support among the tools evaluated here, so these layout managers are not considered in detail.

For me, the choice of the "main" layout manager for my applications comes down to FormLayout vs. GridBagLayout. If I am building forms using pure code with no GUI builder, there is no contest: FormLayout wins hands down. It gives the best results and the highest productivity. If I am using a GUI builder, then the choice is not as clear. I would still prefer to use FormLayout, but there are excellent GUI builders for both FormLayout and GridBagLayout.

2.3 Tool Types

There are different types of GUI builders:

- Full IDE. A full integrated development environment for building the GUI, developing the supporting code, and "wiring together" the GUI with event handling and other related tasks. Examples include the Eclipse Visual Editor, NetBeans, JBuilder, and JDeveloper.
- IDE plug-in. A GUI editor designed specifically to work within an IDE. Examples include Swing Designer, Jigloo, and Jvider.

  Strictly speaking, the Eclipse Visual Editor is also an IDE plug-in, since it is not provided with the base Eclipse download, but must be downloaded separately. However, I put it in the Full IDE category because it is provided by the same "vendor" (eclipse.org) as the IDE itself.
- WYSIWYG editor. A panel builder that gives a "what you see is what you get" editing view. Generally you drop components onto a panel and move them around. After the panel looks right, you go through some code generation or other integration step to make the panel available to your Java code. Examples include JFormDesigner and Foam.
- Non-WYSIWYG editor. A panel builder that helps you build panels by grouping components and setting properties, but without the immediate visual feedback you get with a WYSIWYG editor. These tools usually have a "preview" button so that as you are building your panel, you can view the result without having to integrate the panel into a complete program. Examples include FormLayoutMaker and SpeedJG.
- Pure code. A library intended to ease or improve GUI development, without providing a visual tool. The focus of this article is to find a visual tool, so such libraries are not covered in detail here. Although the pure code approach sounds more primitive than using a visual designer, some programmers feel that it is better than visual development, because the common code can be factored out more easily, and you do not get the code "bloat" that is typical with GUI code generators. An example of a pure code approach is the PanelBuilder class provided with the JGoodies Forms package.

2.4 GUI Definition Storage

Once you have defined your GUI, in what format does the GUI builder save it?

- Pure code. The tool generates Java code to create the GUI, and parses Java code to read it into the visual editor. It does not save any files other than the Java code. Examples are the Swing Designer, the Eclipse Visual Editor, and Jigloo. Sometimes a tool will include special comments in the generated Java code to help it read back the code and display it in the visual editor.
- Code and metadata. The tool generates metadata describing the forms, as well as Java code to display the forms. An example is NetBeans, which saves the form definition in .form files and generates code in .java files. The .form files live in the same directory as the corresponding .java file.

  Tools that use metadata differ on whether they depend on the metadata at design time only, or at design time and run time. For example, NetBeans uses its .form files only at design time; the runtime is pure Java code. In contrast, Foam uses its .gui_xml files at design time and runtime. As another example, JFormDesigner gives you a choice. It creates .jfd files with the form design. You have a choice of whether to deploy the .jfd files and a (royalty-free) runtime library to load the files, or you can choose to generate runtime code that has no dependency on the .jfd files.

3 Recommended Products

My evaluation resulted in me selecting one overall "best" tool for most of my Java GUI work: Swing Designer. I also identified several other excellent tools that I would be happy to use, depending on the requirements of the project in question.

I can summarize these recommended products as follows:

- **Swing Designer**: Best overall and best integrated solution.
- **NetBeans**: Very good overall and best free/open source integrated solution.
- **JFormDesigner**: Best pure panel layout tool.
- **FormLayoutMaker**: Best free/open source pure panel layout tool.
- **Foam**: Most innovative tool.

Detailed information and pros/cons of these and the other tested products are given in the Product Details section.

One common theme that emerged from my evaluation is: "integration is good". I went into the project not particularly caring whether I used an IDE's native GUI builder, an IDE plug-in, or a standalone product. As I spent time with the products, I came to believe that tight IDE integration is essential to attain the highest productivity. GUI application building is more than just panel building: it involves

integrating the panels into the application, and dealing with component properties, event handlers, and the "glue" between application logic and the GUI.

4 Product Summary

This table lists all of the products that I investigated. They are divided into 3 categories:

- Category A. Products that were tested hands-on, and are recommended as my favorites among those tested.
- Category B. Products that were tested hands-on, but during testing were found to be less suitable than those in Category A.
- Category C. Products that were briefly investigated but were not tested hands-on, due to technical or business reasons.

Within each category, products are listed in estimated order of suitability to my requirements. Please don't put too much weight on the "ranking" or ordering of the products in the table. Instead, I suggest that you use my comments as a guide to help you figure out which products are worth investigating, based on your own requirements and preferences.

#	Product	Comments
Category A: Tested hands-on and recommended		
1	Instantiations Swing Designer $199 per developer Related products: SWT Designer ($199), WindowBuilder Pro ($299)	Best overall and best integrated solution. See Product Details.
2	NetBeans 4.0 Free, Open Source (Sun Public License)	Very good overall and best free/open source integrated solution. See Product Details.
3	JFormDesigner 1.0.2 $159 per developer for commercial use	Best pure panel layout tool. See Product Details.
4	FormLayoutMaker rc7 Free, Open Source (BSD license)	Best free/open source pure panel layout tool. See Product Details.
5	Foam 1.2 $150 per developer	Most innovative tool. See Product Details.
Category B: Tested hands-on		
6	Jigloo 3.1.0 (pre-release) Free for non-commercial	Not as polished as the leading products. See Product Details.

	use; $75 per developer for commercial use	
7	JBuilder 2005 Foundation Foundation edition is free (including for commercial use). Developer edition is $500 and Enterprise edition is $3,500.	No compelling benefit over NetBeans. See Product Details.
8	Eclipse Visual Editor 1.0.2 Free, Open Source (Eclipse License)	Not ready for prime time. See Product Details.
9	Jvider 1.7 $69 per developer	Integration is not rich enough. See Product Details.
Category C: Not tested hands-on		
10	IntelliJ IDEA $499	A full IDE with an integrated GUI builder. At design time, uses XY and Grid-based layouts for component positioning. Stores form definitions in XML .form files. Gives the option of generating compiled Java classes for the GUI, or Java source code. The website states "The GUI Designer does not create a main frame for an application, nor does it create menus". It seems strange that you would have to manually create menus and a top-level container such as a JFrame in an integrated product. Appears to use its own layout manager(s) as opposed to standard ones like GridBagLayout. This is probably a good thing for development productivity, but it may cause some concern about code portability (for example, if you decided to change IDEs, or if IntelliJ maintenance became unavailable). On the other hand, I have been informed that the IDEA layout manager is open source (I have not personally verified this). This would certainly ease the concern about vendor "lock in". Also, there is a third-party plug-in that can convert the .form files to Java source code, including the option to use GridBagLayout (again, I have not verified this capability).
11	Abeille Forms Designer Free, Open Source (LGPL license for Designer, BSD license for Runtime)	One of the better open source choices. Supports JGoodies FormLayout. Comparing Abeille and FormLayoutMaker, I gave a slight edge to FormLayoutMaker, but I think Abeille would still be worth investigating.

		I got some feedback stating that Abeille is actually superior to FormLayoutMaker, but unfortunately I do not currently have time to do a hands-on test of Abeille.
12	SwingEmpire FormBuilder Seems to be free, license not clear; site states that there will eventually be a commercial version	GUI builder for JGoodies FormLayout. Not a WYSIWYG editor, but a form builder with preview.
13	Radical Free, Open Source (Apache license for standalone version; GPL for jEdit plugin)	An open source WYSIWYG GUI builder that uses the TableLayout layout engine. Has a dialog that appears to provide full coverage for setting component properties. Generates code. Runs as either a standalone tool or a jEdit plugin. I prefer to use FormLayout instead of TableLayout, so I would tend to select FormLayoutMaker or Abeille over Radical. Other than that, Radical looks like it's worth investigating.
14	SpeedJG $69 per developer	The approach is to build a GUI using XML as the representation. It is not a WYSIWYG editor, you create the XML using their easy-to-use IDE, then click "Test" to see the generated GUI. Looks intriguing and well done, but I did not test because I am more interested in a visual editor.
15	Oracle JDeveloper It is a bit confusing to find pricing information. At the Oracle Store, there is a product called "JDeveloper - Named User Plus Perpetual" listed at $995.	Originally based (circa 1997) on Borland's JBuilder, similarities between the two products can still be clearly seen in the GUI builder (e.g., support for XYLayout). There are many features that make JDeveloper stand out, but from a pure GUI builder standpoint, there is no compelling benefit over NetBeans. Since the JDeveloper GUI builder was originally based on JBuilder, see the comments about JBuilder in Product Details.
16	Ribs Could not find pricing	Looks like it was never officially released (still in preview). Uses absolute positioning and sizes.
17	AbaGuiBuilder Free, Open Source (GPL)	Claims to be "Under Development" but seems to be actively worked on. Both design time and runtime are GPL licensed. Design time GPL is OK, but runtime GPL is a showstopper for me, so I did not test.
18	LayoutBuilder $99 per developer	Appears to support only "standard" layout managers (GridBagLayout, FlowLayout, etc.). Thus it is less interesting than standalone layout tools like

		JFormDesigner that support JGoodies FormLayout.
19	JFrameBuilder $67 per developer	Generates code, seems to be one-way.
20	JBeaver 49 Euros	The site says "A free trial version of JBeaver is downloadable from this website" but I could not find the download link. The screenshot of the Design view looks very nice. Supports JGoodies FormLayout.
21	User Interface Compiler Seems to be free, license not clear	Uses Qt Designer as the designer, then compiles it into (?) XML.
22	BX for Java Pricing not clear, perhaps $998 per developer	Too expensive for me.
23	Visaj $995 per developer	Too expensive for me.
24	JEasy 780 Euros for distributable Pro version	Too expensive for me.

5 Product Details

This section contains detailed information on all the products that I tested hands-on.

5.1 Swing Designer

Instantiations Swing Designer 4.0
Price/License: $199 per developer
Related products: SWT Designer for SWT developers ($199) and WindowBuilder Pro with both Swing and SWT support ($299).

Swing Designer is an Eclipse plug-in. It operates by generating and parsing Java code; there are no separate form definition files. Swing Designer supports standard Java layouts including GridBagLayout and SpringLayout, and has recently introduced support for JGoodies FormLayout.

There are some nice demos on the website.

When building a form in Swing Designer, the workflow is as follows:

- Create a class based on one of the supported visual classes (JPanel, JFrame, etc.)
- Set the layout manager for the panel
- Drop components from the palette onto the panel; drag them around on the panel to position them

- Use the Layout Assistant or other provided tools (described below), set layout properties for the components
- Use the component properties editor to set the visual (e.g., font, color) and code generation (e.g., variable name) properties
- Add event handlers (for example, handling button clicks)

The design view is WYSIWYG, and there is also a preview button to show the designed panel in an example JFrame container.

Swing Designer has excellent support for JGoodies FormLayout. With FormLayout, dropping and moving components is easy and precise. The row/column grid gives you the overall context of the container. As you mouse over the cells in the grid, they are highlighted to show which one you are over, and whether it is valid to drop the component there.

Pasting components is especially nice: when you execute the Paste command, the mouse cursor becomes an arrow with a + sign, and you can click on any valid cell to paste the component. Imagine the common case where you have a new panel and you need to quickly add 10 field labels. Just create one label, copy it, then paste it 9 times, edit the text and you're done. Swing Designer supports in-place editing of the component text (e.g., the labels in this example).

All of the FormLayout constraints/layout properties are supported for rows, columns, and cells. You can edit these properties in several ways, depending on which is most convenient for any given situation:

- Use the Layout Assistant, a popup window that lets you set the row, column, and cell properties for the currently selected component. Because the Assistant is non-modal, you can set the properties for a component, then click on another one and set its properties, and so on. You can also select multiple components to set properties for all of them at the same time.
- Use the Alignment Figures, small buttons that alllow you to quickly set the vertical and horizontal alignment or fill.
- Use the Row/Column Editor, a dialog that shows all the rows or columns in the layout so you can quickly edit them in a tabular fashion.
- Use the context (right-click) menu on row or column headers to insert rows or columns, quickly adjust their properties, or invoke the Row/Column editor.
- Use the component properties editor to set the layout constraints.

Swing Designer also has good support for GridBagLayout, although I did not spend as much time testing it because I prefer to use FormLayout. When you drop or move a component, the designer shows you the existing GridBag cells, plus new cells where the component can be dropped. The new cells are not only along the edges of the layout, but in between every existing row and column. This makes layout very productive because you can add new rows and columns on the fly, without having to manually "make room" by moving existing components.

Components are created as local variables by default. You can tell Swing Designer to create a member variable for a given component by using the "Expose component" context menu option. Or you can set a preference to automatically create every component as a field.

Creating event handlers is simple: for example, just double-click on a button to create its event handler and view/edit the source. More generally, you can right-click any component, select the Implement context menu, and then choose any event supported by that component to implement its handler. You can set a preference to specify the naming convention for generated event handler methods.

Swing Designer has good support for menus and toolbars. I was able to quickly add these elements to the main JFrame of my test application, and automatically create stubs for their actionPerformed handlers.

When to use Swing Designer
Swing Designer hits the mark perfectly for me: it is an Eclipse plug-in with tight IDE integration; it has a fantastic visual designer; it has excellent support for JGoodies FormLayout; and it is reasonably priced. Therefore, Swing Designer is my first choice for my GUI work.

Pros

- The best visual designer (fast, flexible, and full-featured) of any IDE or IDE plug-in that I tested.
- Excellent JGoodies FormLayout support, which leads to higher development productivity and better user interfaces.
- Very broad and deep support for many different layouts and components.
- Seamless Eclipse integration: component properties, code generation, etc.

Cons

- The first time you launch the designer in a given Eclipse session, it is rather slow to start (10 or 15 seconds on my machine). This is not much of an issue because once the designer is started, it is very fast - I never felt that it lagged behind during visual editing.
- I did encounter some rough edges (bugs and other issues). However, the vendor provided excellent support, often with a same-day or next-day patch or workaround.
- The documentation and website could use some polishing up - they don't reflect the completeness and quality of the product.

5.2 NetBeans

NetBeans 4.0
Price/License: Free, Open Source (Sun Public License)

NetBeans is a competitor to Eclipse as a full featured open source IDE. I am an avid Eclipse user, and I never considered trying NetBeans before this project. People have strong feelings about which IDE is best, but this article is not a general IDE discussion.

In this article we are focusing on the GUI-building capability of the tools, and in this respect, NetBeans is clearly superior to Eclipse. My comments (as with all the product details in this section) apply to the specific versions that I tested: NetBeans 4.0 vs. Eclipse Visual Editor 1.0.2 running in Eclipse 3.0.1. These were the latest production releases available at the time I tested.

When building a form in NetBeans, the workflow is very similar to the workflow described above for Swing Designer. The main difference is how the components are positioned on the panel (described in detail below). Also, what Swing Designer calls the Layout Assistant is known as the Layout Customizer in NetBeans.

The design view is WYSIWYG, and there is also a preview button to show the designed panel in a mock JFrame.

In NetBeans, when you drop a component on the form, you cannot control the original placement of the component - no matter where you drop it, it will go into a new cell. This is not as friendly as (for example) Swing Designer or Eclipse Visual Editor, which allow you to place the component in any cell. There is a similar issue when you copy and paste a component: the pasted component sits in the same cell as the copied component. So, after dropping or pasting, you then need to use the GridBag customizer to place the component where you want it. This is a major hindrance to productivity because you need to switch between the design view and the customizer view for each component you need to position. It is true that you can drop or paste multiple components (say 5 of them) in design view, and then go to the customizer and move those components. But this is still not ideal because it would be better to drag-and-position or paste-and-position in the same view.

On the other hand, the NetBeans GridBag customizer is much better than the one in the Eclipse Visual Editor. It has more properties that you can set and is easier to use.

At one point, I had a problem when moving components in the customizer and going back to the design view - the layout was garbled with overlapping components. But then I just needed to expand the size of the container (a JPanel) so that everything fit, then the layout looked correct again. This seems to be a common issue (not specific to NetBeans) with the GridBagLayout at both design and run time: you need to make sure your container is big enough. The way that I deal with this at runtime is to calculate the minimum size needed to properly display each panel, and then restrict the main JFrame so that the user cannot resize it smaller than this minimum.

A technique described in the NetBeans documentation is to start building a form using the AbsoluteLayout, which allows positioning of each component. Then convert to GridBagLayout for proper resizing behavior. I found this approach more frustrating than just building with GridBagLayout from the start.

The NetBeans GUI builder is not as powerful as some of the other products, but its tight integration with the overall IDE makes it convenient and productive to use.

When to use NetBeans

If I wanted to build a full application (panels, menus, toolbars) with an integrated open source IDE, without adding any external plug-ins, NetBeans would be the best choice.

Pros

- Excellent integration betwen the form designer and the overall code development process. In particular, setting properties, code generation options, and event handling are very fast, easy, and complete.
- Good designer for GridBagLayout, although not as good as Swing Designer or Eclipse Visual Editor.
- The best layout customizer for GridBagLayout.

Cons

- My perception is that NetBeans has been losing the open source market share / mind share IDE battle with Eclipse (on the other hand, there is some evidence of a NetBeans resurgence). My guess is that future releases of the Eclipse Visual Editor may provide much stronger competition. Still, I'm not worried about getting "locked in" with NetBeans, because it is open source, and the generated code is standard Java.
- Does not support FormLayout or other replacement for GridBagLayout (i.e., you are pretty much forced to use GridBagLayout for complex forms if you want to use NetBeans' integration).
- Cannot move components in design view using drag and drop, need to go into customizer to move.
- No "undo" or "cancel" in GridBagLayout customizer.

5.3 JFormDesigner

JFormDesigner 1.0.2
Price/License: $159 per developer for commercial use

JFormDesigner is a very polished and professional tool. It does an absolutely outstanding job of helping you layout panels. With the excellent FormLayout support, it is almost trivial to drag your components into place and get your layout looking good. However, once you save your files (.jfd form files and .java code files) from JFormDesigner, you are "on your own" - there is no integration with development environments like Eclipse or NetBeans.

There is a nice Flash demo available on the website.

JFormDesigner supports standard Swing layout managers plus the third-party FormLayout and TableLayout. The support for JGoodies FormLayout (layout manager) and JGoodies Looks (look and feels) is extensive. JGoodies FormLayout is my favorite layout manager, and JGoodies Looks is my favorite look and feel, so support for these is a big plus. You can build a really professional looking application with JFormDesigner, JGoodies FormLayout, and JGoodies Looks.

In JFormDesigner, you create your form by dragging components in the design view. The tool saves the form definition in .jfd files which live alongside Java code. Then there is a generate code step that can translate the .jfd files to Java code. You have a choice of whether to deploy your forms as .jfd files, which are parsed at runtime,

or as .java code. If you use .jfd files, you also need to deploy the JFD parser JAR file with your product, however this is royalty-free.

Because the form editor (JFormDesigner) and code editor (Eclipse or vi or whatever editor you use) are separate, you have to be careful about collisions when saving files. The documentation says: "As long as you follow the following rule, you will never have a problem: Save the Java file in the IDE before saving the form in JFormDesigner."

When to use JFormDesigner
If I had to create a large number of panels quickly and productively, and I was not concerned with IDE integration, I would use JFormDesigner with FormLayout.

Pros

- Supports standard layout managers (including GridBagLayout), plus FormLayout and TableLayout.
- Has extensive additional support for JGoodies Forms (FormLayout and related classes) and Looks (look and feels).
- The best visual designer for FormLayout.

Cons

- No IDE integration.
- No event handling code generation.
- No support for externalizing strings (e.g., label or button text).

5.4 FormLayoutMaker

FormLayoutMaker rc7
Price/License: Free, Open Source (BSD license)

FormLayoutMaker is a non-WYSIWYG designer specifically for the FormLayout layout manager. You add components to a grid, which does not look like your final panel - it is just a visual representation of the position of the components within the grid. After you drop your components, you can hit the preview button to see what the actual panel will look like. Once your panel is looking good, you can save the XML layout file and the Java code.

Another open source panel builder worth looking at is Abeille Forms Designer; please see the comments about Abeille in the Product Summary.

When to use FormLayoutMaker
If I wanted an open source tool to help me build panels using FormLayout. FormLayoutMaker is not as polished as JFormDesigner, but it will probably hit the sweet spot for a lot of open source developers.

Pros

- Good visualization of the structure of the layout with the grid view.
- Easy preview for visualization of the actual layout.

- Good support for FormLayout (although the support is not as advanced as in JFormDesigner).

Cons

- No IDE integration.
- No support for setting properties, event handling, etc.

5.5 Foam

Foam 1.2
Price/License: $150 per developer

Foam is a delight to use - it feels different than every other tool I evaluated. As I was using it, I was thinking "this is easy!" and "this is fun!". It is probably the fastest for laying out a panel of any tool that I tested. You drag a component from the palette onto the design view, and it shows you crosshairs and anchor points so you can align it and "snap" it to certain predefined points, like the edge or center of other components. This was the most natural feeling designer I tried.

There are some excellent Flash demos available on the website.

The form design is saved in an XML file. A runtime JAR (royalty-free) parses the file and renders the form.

My concern with Foam is the proprietary nature of the runtime engine: if the vendor decides to no longer maintain the product, you are stuck with the latest release. This is troublesome when compared to other products:

- For open source products, this is not a concern, because you can (in theory at least) fix the bug yourself, or try to organize community support with other users.
- For other commercial products, like JFormDesigner, it is true that the designer is proprietary. But the runtime is "open" in the sense that JFormDesigner can generate standard Java code that does not refer to the form design metadata at runtime. Therefore, in the worst case, you could hand-edit the generated code if support for the tool was no longer available.

These options are not available with a proprietary runtime engine.

When to use Foam
If I had to quickly prototype a set of panels, for example for an application design proposal, Foam would be the quickest way to do so.

Pros

- Probably the quickest for prototyping. Absolutely slick environment.
- Makes it simple to develop very structured and elegant panels without knowing details of layout management.

Cons

- Proprietary runtime.
- No IDE integration.
- No copy and paste of components in the design view.

5.6 Jigloo

Jigloo 3.1.0 (pre-release)
Price/License: Free for non-commercial use; $75 per developer for commercial use

Jigloo works by generating and parsing Java code ("round trip code generation"). It is an Eclipse plug-in, not a standalone product.

Jigloo supports both Swing and SWT. For Swing, it supports standard layout managers, plus Jigloo AnchorLayout and JGoodies FormLayout. For SWT, it supports standard SWT layouts including FormLayout (the SWT FormLayout is not related to the JGoodies FormLayout, except that they unfortunately have the same name).

Visual editing was OK, but not as smooth and polished as Swing Designer or JFormDesigner. It was hard to lay things out properly. The grid only appears when mousing over the panel, and I found this grid harder to work with than (for example) Swing Designer or the NetBeans GridBag customizer. JGoodies FormLayout support is not as advanced as Swing Designer or JFormDesigner; for example, Jigloo does not automatically insert spacer rows and columns.

5.7 JBuilder

JBuilder 2005 Foundation
Price/License: Foundation: free (including for commercial use); Developer: $500; Enterprise: $3,500

The JBuilder GUI building support is roughly on a par with NetBeans and Eclipse. Since it does not offer any compelling advantage over the open source alternatives, I did not spend a lot of time with it.

The JBuilder docs suggest starting your layout with the included XYLayout, and then after you get it looking right, convert it to GridBagLayout. Just as with NetBeans' AbsoluteLayout, I found it simpler to just start with GridBagLayout from the beginning.

Borland offers extensive documentation on GUI building in JBuilder. Even if you don't use JBuilder, it is worth reading this documentation, especially if you are using GridBagLayout. Here are some recommended links:

- Using layout managers
- GridBagLayout
- Using nested panels and layouts

5.8 Eclipse Visual Editor

Eclipse Visual Editor 1.0.2
Price/License: Free, Open Source (Eclipse License)

On the Eclipse website, the overall Eclipse project is described as: "a kind of universal tool platform - an open extensible IDE for anything and nothing in particular". In that spirit, the Eclipse Visual Editor is described on the website not as a GUI builder, but rather as "a framework for creating GUI builders". However, the Visual Editor does come with "reference implementations" of Swing and SWT GUI builders.

Eclipse VE uses Java code generation/parsing. It is slow to start up. Sometimes it seems reasonably fast when editing. But other times, a simple task (like clicking on a component in design view) could take 5 or 15 seconds. This can be very frustrating.

In Eclipse VE, the process of dropping components is very nice. You have a grid in the design view and you can drop the component into an existing cell, or into a new cell between any existing cells or on any edge of the grid. The VE nicely highlights the existing or new cell where the component will be dropped. However, once you have components in the cells, it is not easy to tell which cell you are working with.

Property and constraint editing is not as smooth as with some other products. For example, I found it hard to set minimum widths and spacing. The GridBagLayout customizer is not as advanced as the one in NetBeans.

JSeparator is not on the control palette. I added one using the generic add JavaBean feature. The designer accepted it, but I could not see it in the visual editor or the running application.

I had a couple of situations where the layout in the design view did not match the runtime. Also, somehow the layout got garbled, with components overlapping each other (and I could not find a simple fix like I did in NetBeans).

The bottom line is, I found Eclipse VE to be not acceptable for these reasons:

- Response time inconsistent, sometimes frustratingly slow
- Visual design process not as clear as with other products tested
- Layout not reliable (design time not matching runtime, or layout garbled)

In fairness, Eclipse VE is a relatively new product, and I look forward to future versions being much improved.

5.9 Jvider

Jvider 1.7
Price/License: $69 per developer

Jvider offers both standalone and Eclipse plugin versions. The same license can be used for both. Jvider focuses on being a lightweight, flexible solution instead of a

total solution. For example, not all properties can be set through a property editor, you need to edit code. It focuses more on the layout of components than the details of each component. Jvider uses GridBagLayout.

I found that Jvider was too lightweight for me, it did not provide enough properties access. Since it is using GridBagLayout, I would prefer to use NetBeans which offers a richer and more integrated experience.

6 Wish List

As I mentioned earlier, Swing Designer has the best combination of features for my needs. Many of the other products have great features but are lacking in various ways. Hopefully over time these products will continue to evolve and improve. For example:

- Add design view component positioning (and ideally, FormLayout support too!) to NetBeans.
- Add IDE integration to JFormDesigner.
- Supply more open licensing for Foam runtime.
- Improve Eclipse Visual Editor robustness.

7 Example Code

For an example of a GUI built with Eclipse, Swing Designer, and JGoodies FormLayout, see my JMailSend project.

For an example of a GUI built with NetBeans and GridBagLayout, see my HashGUI project.

For an example of a GUI built with hand coding (no GUI builder) using FormLayout, see my PWKeep project. I wish that I had thought to look for a GUI builder when I was writing PWKeep - it would have made the form creation process much smoother.

8 Revision History

- 1.4 (06-Apr-2005): Minor copy edits.
- 1.3 (06-Apr-2005): Added hands-on review of Swing Designer. Adjusted comments on some other products based on the features of Swing Designer. Updated IntelliJ IDEA listing with new information about layout manager portability. Added some comments about how to interpret the "rankings" of products in the table. Added link to JMailSend project.
- 1.2 (26-Mar-2005): Added link to JavaLobby discussion thread. Moved Swing Designer up in the rankings and added a few comments due to new release that supports JGoodies FormLayout. Added IntelliJ IDEA to Product Listing. Added link to SGLayout. General copy editing.

- 1.1 (24-Mar-2005): Added some layout managers to listing. General copy editing.
- 1.0 (19-Mar-2005): Initial revision.