Answer on Question #48635, Engineering, SolidWorks | CosmoWorks | Ansys

Task: Explain all the steps of running the beanbox.

Answer:

The *BeanBox*, from Sun, eases the development of JavaBeans by providing a test container for the components. The BeanBox can test that the code you wrote for your bean works properly inside a container.

Running the BeanBox

To run the BeanBox from the command line, change directories to the directory in which you installed the BDK. (From now on, we'll call this the "BDK directory.") Then switch to the beanbox subdirectory. This directory contains two files, *run.sh* (for Unix) and *run.bat* (for Windows). Running the batch file appropriate for your operating system will start the BeanBox.

If you're a Windows user, you can create a batch file called BEANBOX.BAT and place the following lines in it:

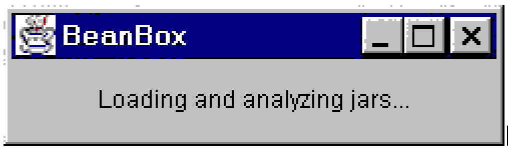
```
@echo off

cd BDK-directory\beanbox

run.bat
```

(Be sure to replace *BDK-directory* with the name of the BDK-directory.) Users of Windows 95 or Windows NT can even create a shortcut to this batch file, and run the BeanBox by simply clicking the shortcut's icon.

Upon running the BeanBox, a small window will appear, saying Loading and analyzing jars....

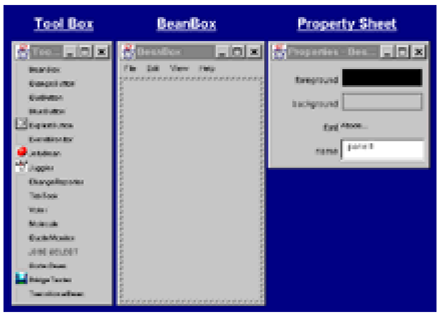


Loading and analyzing jars...

Loading JARs

At this point, the BeanBox is reading the JAR files in the jarssubdirectory of the BDK-directory. A JAR file is a ZIP archive file containing class files (including beans), GIF files, and other resource files the classes may need, plus an optional *manifest* containing structured information describing the contents of the JAR. You can create your own JAR files and put them in the jars directory, and the BeanBox will include them in the ToolBox. See Resources below for links to finding out more about JAR files.

After a few moments, the small JAR file window disappears, to be replaced by three windows, as seen in the figure below:



The BeanBox windows

These three windows are:

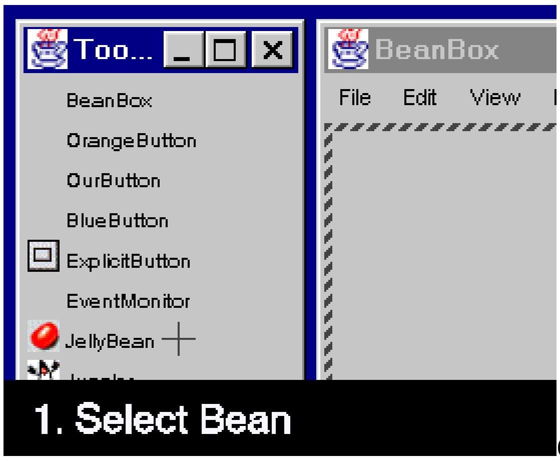
- On the left, the *ToolBox*. This window contains all of the beans that are available for adding to the BeanBox.
- In the middle, the *BeanBox*. This window contains the beans we're working on.
- On the right, the *Property Sheet*. This window contains property editors for all of the properties of the currently selected bean. (In Figure 2, the currently selected bean is the BeanBox itself, as evidenced by the crosshatched box around the perimeter of the BeanBox.)

The BeanBox currently is in *design mode*, meaning that you can select beans from the ToolBox, place them on the BeanBox, change their properties by using the Property Sheet, and manipulate their behavior with the items on the menu across the top of the BeanBox.

Since we're in design mode anyway, let's add a bean to the BeanBox and see what we can do with it.

Adding beans to the Box

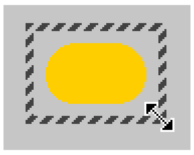
Adding a bean to the BeanBox is as simple as clicking on the bean's label in the ToolBox and clicking in the BeanBox at the position where you want the bean to appear. When you click the bean's label, the cursor changes to a thin pair of crosshairs. Clicking anywhere on the BeanBox creates a new bean, centered around the point where the click occurred. (See the figure below.) Note that nowhere is there any indication of what bean has been "picked up" by the crosshairs.



Choosing and adding a bean

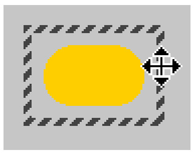
You can also graphically move and resize the beans, though to do so, you need pretty good control of your mouse. At first, it's a little hard to find the "hot spots" where a change in the cursor shape indicates that the bean can be moved or resized. This is one of the BeanBox's many small user interface quirks that make it more difficult than necessary to use.

To resize a bean, position the cursor near the hatched border next to the corner you want to drag (as seen in the figure below). The cursor will change to a diagonal line with arrows.


Resizing a bean

Hold down the left mouse button and drag, releasing when the hatched box indicates the component size you desire. Some beans (like the JellyBean) won't allow you to resize them: When you release the mouse button, the box simply snaps back to its original size. The only way to tell which beans behave this way is by experimenting.

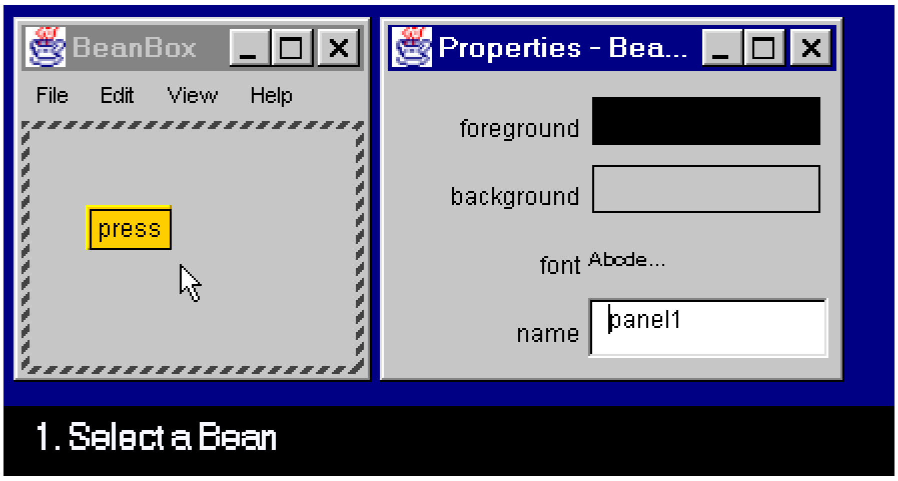
Moving a bean works similarly, except that the hot spot is along the edges of the hatched box instead of at the corners. The cursor changes to crosshairs with arrows, indicating that you can click and drag the bean around.


Moving a bean

It's easy enough to place beans in the BeanBox; unfortunately, hitting the backspace or delete keys won't delete the bean as you might expect. Instead, select the bean and delete it using the *Cut* item on the *Edit* menu. Small details like this make using the BeanBox a bit awkward at first.

Changing a bean's properties

It's simple to change properties of beans in the BeanBox. First, select the bean whose properties you want to change by clicking it with the mouse. You'll notice that the Property Sheet window will change to reflect the available writable properties of the bean. You can change the selected bean's properties inside the Property Sheet. The figure below demonstrates how to change the properties of a JellyBean.



Changing a bean's properties

Every type of property has an associated "property editor," which is a widget used for editing a particular property type. Sometimes these are widgets embedded in the Property Sheet; other times (such as more complex properties like Color and Font), the user uses a small pop-up dialog box to change the property. You can write your own property editors for your own types (though that is beyond the scope of this article).

Dissecting a bean

The BeanBox can produce a report of any bean's properties, methods, and events. For example, we've been looking at a very simple bean called a JellyBean, which has just a couple of properties and a few methods -- or so it seems. To "dissect" the JellyBean and find out what services it provides, select the bean and then select the *Report...* item on the *Edit* menu. Here we come across yet another BeanBox annoyance: The report is printed to the standard output of the java interpreter running the BeanBox. If the window from which the BeanBox was run is minimized, it looks as if the report is never run. The only way I've found to capture the output of the report is to change the line in run.sh (or run.bat) that runsjava to redirect its standard output to some file:

```
...

java sun.beanbox.BeanBoxFrame > beanbox.txt
```

If you take a look at the output of the report, you'll notice a surprisingly large number of methods and attributes for such a simple bean. This is because the reporter uses the Java *reflection* mechanism, the new addition (as of JDK 1.1) to the core API that allows Java programs to analyze class files. The reflection mechanism is very thorough: A class file has no secrets left once java.lang.reflect is through analyzing it. Most of the methods you see in the report output come from classes that the bean inherits (often especially from java.awt.Component).

Hooking 'em up

Beans communicate by passing around Event objects, which Are derived from java.util.event. The event type indicates what type of information the Event contains. A bean's *event set* is the set of events that a bean can process -- events that make sense for that bean.

It's actually possible to set up relationships between beans without writing any code. Although we won't go into the details here, the process is straightforward. The BeanBox knows how to analyze which Events "source" beans can produce, as well as which Events "target" beans understand. It then creates a small custom class (called an *event adaptor*) to route messages from Event sources to their targets.

Using the BeanBox, follow these instructions to "hook" two beans together:

1. Place a Molecule bean and an ExplicitButton bean in the BeanBox.
2. Select the ExplicitButton bean, since it's the event source.
3. Select *Edit* -> *Events* -> *button push* -> *actionPerformed* from the menu. The BeanBox has determined that the ExplicitButton class can produce this event (it does so when the button is pushed).
4. You will notice that a red "rubber band" follows the cursor around. Left-click the mouse on the Molecule bean. This identifies the Molecule bean as the event target (the receiver of the Event).
5. Another menu comes up: This time, it's the list of methods you can call on the target bean when the source bean produces an actionPerformed event. Select "rotateX" and hit "OK."
6. A window appears with the label Generating and compiling adaptor class. The BeanBox is creating a class that calls rotateX() on the target whenever theactionPerformed occurs on the ExplicitButton. It then associates the source and destination objects with this "adaptor" object.
7. Now, click the ExplicitButton bean. The adaptor class detects the actionPerformed call, and calls rotateX() on the Molecule, and the Molecule rotates!

Saving and loading beans

You can save your little applications in the BeanBox by using *Save...* on the file menu. *Load...* will let you reload any files you've saved in the past. Not only are the positions, sizes, and so on, of your beans saved, but all of the beans' *internal state* is saved, as well. This means that if you save the "rotating molecule" in the previous example to a file, when you load it again, the molecule will be in the same position as when you saved it.