

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa

Melksham

Wiltshire

UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

Java Beans

By using set naming conventions for your methods, you can write a Java class that has its functionality defined by the method name. Development environments used by the application programmer who's using your class can then examine your class and provide context-sensitive menus, help, icons etc.

<i>Beanboxes</i>	4
<i>Indexed, bound and constrained properties</i>	5
<i>Auxiliary classes</i>	5
<i>Sample Beans</i>	6
<i>Tools</i>	8

What are beans?

By their nature, applets and graphic classes can't be developed in a perfect world. They don't always function exactly as you want the first time. And whilst other people's classes have a rigid interface defined by object-oriented methods, they may not do what you want, or have some surprising behaviours or bugs in them.

Large chunks of code and classes can cause something of a delay in loading the application or applet. That can be frustrating, especially when some of the code loaded might never be used. Examples:

- A configuration or preferences menu which, once set and stored, hardly changes.
- Interpretation of incoming data where different clients supply the data in different formats.

These are some of the shortcomings that Java beans are designed to overcome.

A Java bean is not a class!

Any object which conforms to certain rules may be used as a Java Bean, and indeed all the AWT components in Java 1.1, can be used as beans if you wish.

Many beans are AWT components, i.e., things that are visible within your applet or window, but that's not vital. There are useful things that can be done with beans even if they are not normally visible.

As far as the developer of a bean is concerned, beans consist of a number of elements:

- The `.java` and `.class` files for the bean itself
- a `BeanInfo` class
- a Property Editor
- a Bean Customizer

(You don't need to define all of them!)

The developer of applications that use your beans may find whichever of these you provide useful.

The user of an application will not need more than the `.class` file (packaged in a jar).

2.1 Beanboxes

From a developer's viewpoint, it's not always easy to debug a graphics program. You yourself may have struggled with graphics layouts, for example.

Sun's "Bean Development Kit" (an extra product and not part of the standard Java distributed with Solaris 2.7, for example) provides a beanbox in which your beans can be run/tested/examined/alterd very much in the way a debugger might be used in traditional languages to examine and, if necessary, set variables as the thing (class or bean in this case) is run.

We change and examine variables in a traditional debugger; whereas, in a bean we can examine and change properties and see the effect on a bean. This is when we come to the property editor and the first rule that must be applied as you write a bean:

- You must provide `set` and `get` methods for each of the properties which you wish to handle.

AWT components can function as beans. Look at the definition, for example, of `java.awt.GridLayout`. It includes:

```
getColumns    setColumns
getHGap      setHGap
getRows      setRows
getVGap      setVGap
```

In other words, it can function as a bean with four properties.

To prepare a bean for use in a beanbox:

- You must package it up in a jar file – a Java Archive file – similar to a tar file, except that it uses internal compression and it includes a list of contents, i.e. a manifest, in the file.
- You must also ensure that the following line is present for the class within the manifest file by providing a manifest file yourself to jar:

```
Java-Bean: True
```

As a developer running beans within a beanbox, you'll be given a beanbox window, and as you run your bean or as you handle events ...

Oh! Your bean must generate events in the same way that the Java 1.1 AWT does by adding and removing `event listener` objects for the event.

As events are generated or as the bean is run, the results appear in the beanbox window. Further additional information is also presented within that window so that you can see what is happening. A properties window can also be present so that you can examine or change the properties. This might be a default property editor, or whatever you provide yourself.

The procedure for installing a bean in a beanbox will vary depending on what beanbox you use. The interface is publicly defined and in theory anyone with the knowledge could write his own beanbox.

In the case of Sun's BDK, just copy the bean's JAR file into the jar's subdirectory within the BDK directory, and you'll be offered the bean on a menu of beans each time you run the beanbox.

2.2 Indexed, bound and constrained properties

The properties we saw within the `GridLayout` were all simple settings, but the JavaBean API also supports three other types of properties:

- **Index Properties**, for a property which is defined with a whole array of values.
- You must provide methods to `set` and `get` individual elements of the array and methods to `set` and `get` the whole array.
- **Bound Properties**, which send out notification events when their values are changed.
- **Constrained Properties**, which also send out notification events. In the case of constrained properties, the listener is allowed to veto the proposed setting.

2.3 Auxiliary classes

For use in the beanbox. Possibly also used by the application developer who uses your bean. Almost certainly not included in the final distribution to end users.

- The **BeanInfo** Class

You must name this class the same as your Bean class, followed by the word **BeanInfo**.

Thus, if your main class is called `fieldfare`, then the `BeanInfo` class should be defined as:

```
public class fieldfareBeanInfo extends SimpleBeanInfo{
```

Within the `BeanInfo` class, you should include:

- A **BeanDescriptor** object which includes a reference to the Customizer class for the object
- You provide a method `getBeanDescriptor`, which returns an object of type **BeanDescriptor**.
- An Icon to represent the bean
- You should provide a method `getIcon`, one integer parameter, which returns an object of type `Image`.
- A list of supported properties, each with a short description

- You should provide this through a `getPropertyDescriptors` method, which returns an object of type `PropertyDescriptor`.
- A list of methods supported by the bean
- You provide a `getMethodDescriptors` method.
- A method which returns the most commonly altered property, known as the default property
- You provide `getPropertyIndex` which returns an integer pointer to the array of properties.
- References to property editors
- If you are going to define your own property editor (a text string editor to change properties is provided by default) you must provide a `getCustomEditor` method.
- If you are defining a `getCustomEditor`, you must also define `supportsCustomEditor`, returning a boolean true.

None of these methods or objects are going to be referred to by your application or super-class. Rather, they are there as tools for the user of the beanbox and they will be referenced by the beanbox within which your bean is being tested and used.

The environment for which fully implemented beans will be of particular use are for larger Java projects, or for elements which are going to be offered for commercial gain where the developer wishes to give his customer every encouragement to use the product.

It is unlikely that you'll want to package all your one-off classes as beans, although you may stick to the rules so that they may be packaged later.

Of the various text books we have on this course, only two of the ten-or-so actually cover beans. Although the topic is a huge one (afterall, there is a complete book on the topic in the *O'Reilly* series!), it is not (yet?) what we would describe as core Java. Remember, whilst Sun ship the Java Development Kit with their operating system as standard, they do not (yet?) ship the bean development kit.

Quoting from one of the course texts published in September 1997:

"The Beanbox tool shipped ... has a number of shortcomings. In part this is due to the fact that the BDK is a new technology ... It is also because beanbox is intended as a test environment, not an actual programmer's tool."

For these reasons, we are not yet including a larger section on beans on this course, nor talking through nor providing examples of Property Editors or Customizers. Yes, beans may well become important in the future and you should bear in mind the following:

Beans work through introspection. In other words, the beanbox uses information you provide and its own abilities to look inside your classes to provide the interfaces and facilities it does.

Thus, while the name of the class that you package as a bean is not important, the names of the methods you use and their calling sequences most certainly are.

We recommend that you use standards such as the following for your classes so that they are beanable later:

- Have a no-argument constructor
- `getSummat` and `setSummat` to access properties
- Boolean properties (also provide `isSummat`)

2.4 Sample Beans

As you've seen, you can use the bean standards even if you're not going to be using a beanbox, and to do so is a good idea. It means that users of your classes can tell from the method names and calling structures how they're going to work.

In order to help you with the structure, here's a template for a Java bean:

```

//Title:      Javabeandemo
//Version:
//Copyright:  Copyright (c)2004
//Author:     Graham Ellis
//Company:    Well House Consultants
//Description: Sample shell for a Java bean

package jib;

import java.awt.*;

public class Bean1 extends Panel {
    BorderLayout borderLayout1 = new BorderLayout();
    private String sample = "Sample";

    public Bean1() {
        try {
            jbInit();
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.setLayout(borderLayout1);
    }

    public String getSample() {
        return sample;
    }

    public void setSample(String newSample) {
        sample = newSample;
    }

    public static void main(String[] args) {
        Bean1 bean1 = new Bean1();
    }
}

```

And here's a beaninfo class to go with it:

```

//Title:      Javabeandemo
//Version:
//Copyright:  Copyright (c)2004
//Author:     Graham Ellis
//Company:    Well House Consultants
//Description: Sample shell for a Java bean

package jib;

import java.beans.*;

public class Bean1BeanInfo extends SimpleBeanInfo {
    Class beanClass = Bean1.class;
    String iconColor16x16Filename;
}

```

```

String iconColor32x32Filename;
String iconMono16x16Filename;
String iconMono32x32Filename;

public Bean1BeanInfo() {
}

public PropertyDescriptor[] getPropertyDescriptors() {
    try {
        PropertyDescriptor _sample = new PropertyDescriptor("sample", bean-
Class, "getSample", "setSample");
        PropertyDescriptor _age = new PropertyDescriptor("age", beanClass,
"getAge", "setAge");
        _age.setDisplayName("age");
        _age.setShortDescription("age");
        PropertyDescriptor[] pds = new PropertyDescriptor[] {
            _sample,
            _age,};
        return pds;

    }
    catch(IntrospectionException ex) {
        ex.printStackTrace();
        return null;
    }
}

public java.awt.Image getIcon(int iconKind) {
    switch (iconKind) {
        case BeanInfo.ICON_COLOR_16x16:
            return iconColor16x16Filename != null ? load-
Image(iconColor16x16Filename) : null;
        case BeanInfo.ICON_COLOR_32x32:
            return iconColor32x32Filename != null ? load-
Image(iconColor32x32Filename) : null;
        case BeanInfo.ICON_MONO_16x16:
            return iconMono16x16Filename != null ? load-
Image(iconMono16x16Filename) : null;
        case BeanInfo.ICON_MONO_32x32:
            return iconMono32x32Filename != null ? load-
Image(iconMono32x32Filename) : null;
    }
    return null;
}
}

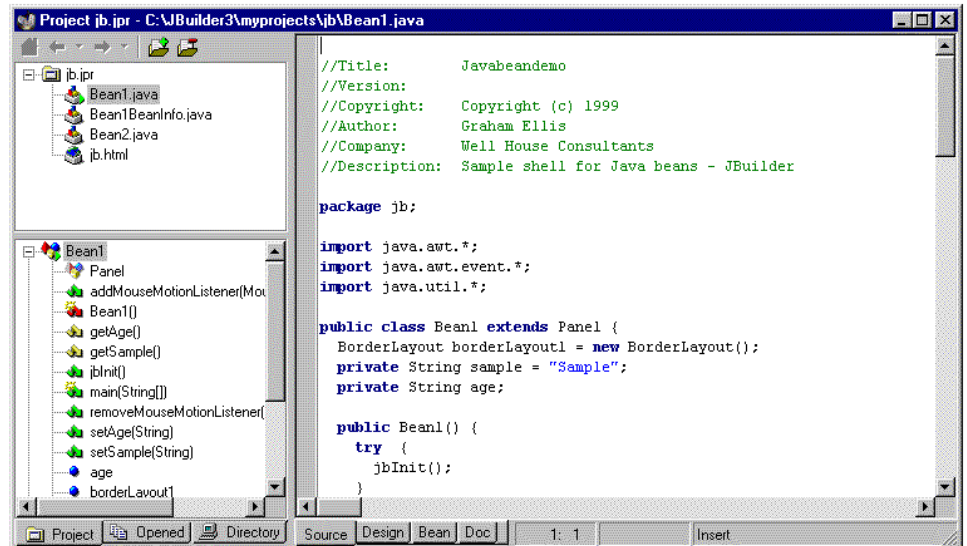
```

2.5 Tools

It is, though, very easy to make an error and add in something nonstandard if you try to write beans using just a text editor and the JDK; you may well want to use a development environment.

The following frames are samples from Borland's JBuilder (version 3), which includes bean creating and validation tools.

Figure 1 JBuilder's point-and-click interface

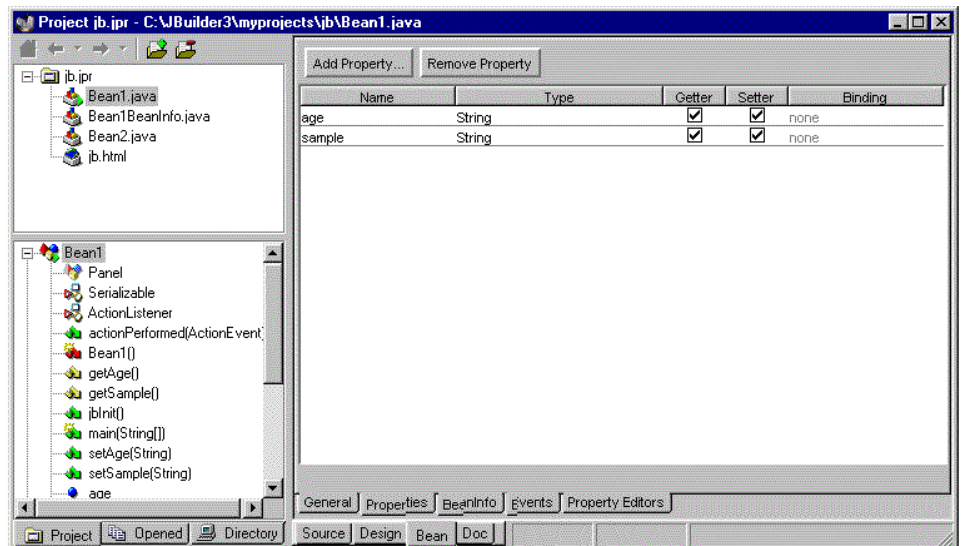


The first example shows the source code of a bean created by the "bean express" wizard. As many beans are graphic-based, we've chosen to extend a Panel (part of the Abstract Windowing Toolkit) with this bean, though we could have simply extended an object, or extended a swing class had we wished.

You'll notice that JBuilder includes a custom editor which highlights syntax, controls that let you add pseudo documentation comment, etc; scarcely a "beans" topic in itself, but something you might like to explore further.

Using a tool such as JBuilder, you can go add and examine properties using a point and click interface, adding in real code via the source window or by selecting from pulldown menus.

Figure 2 JBuilder programmer-specified properties



Also shown in Figure 2 is an example of the programmer-specified properties in our sample bean.

Because the bean we've created in this example extends an AWT class, there are many other properties which are inherited. JBuilder can even go through and tell us details of those properties, as shown in Figure 3.

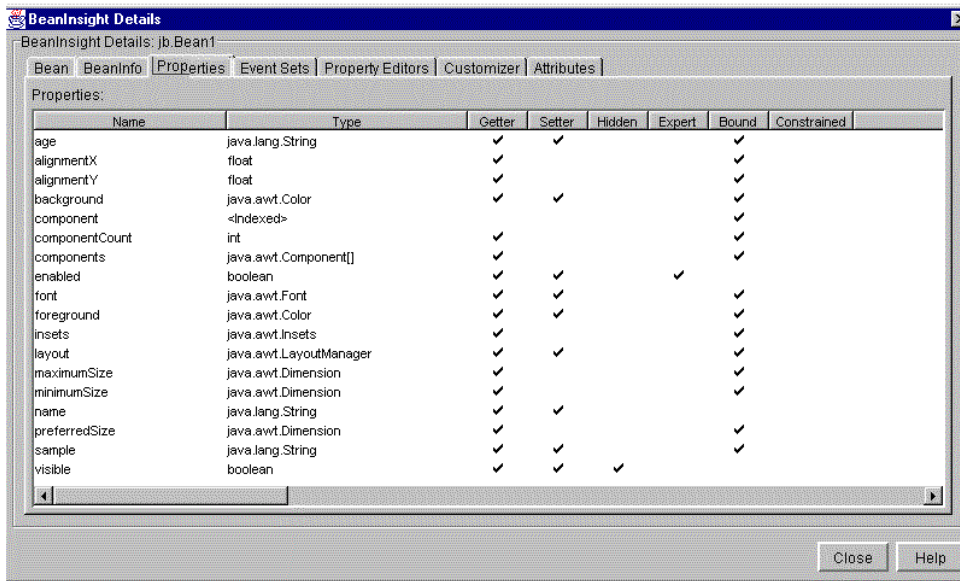


Figure 3 Details of JBuilder properties

Exercise

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.