© Softwaresysteme. Alle Rechte vorbehalten.

Overview

JavaBeans: Introduction

The JavaBeans technology is the **component architecture** for the Java platform.

It defines Java software components and how they fit together.

New applications can be build by using components.

Platform independence:

- Beans are created according to the JavaBeans API specification.
- Beans run in any environment which supports Java.

Platform integration:

- Beans can bridge to native component models,
  e.g. ActiveX, OpenDoc, LiveConnect.
  Software components that use JavaBeans APIs are thus portable to containers including Netscape, Internet Explorer, Visual Basic, Microsoft Word, Lotus Notes, etc.

Definition: “A JavaBeans component is a reusable software component that can be visually manipulated in builder tools.”

What are JavaBeans?

JavaBeans comprises

- a component model for Java to create applications by combining components
- an API that describes the specification of the JavaBeans component architecture
- an extension of the Java platform that adds new levels of flexibility and reuse

Bean:

- A reusable software component.
- Examples:
  - GUI widgets.
  - Non-visual Beans (e.g. database connector)
  - Applications.

Difference between classes and Beans:

- A Bean is a class or a group of classes which follows the naming convention of the JavaBeans API. This allows the introspection process, in which the Bean is analyzed.

Plugging Beans together

Suppose you have the following Beans:

- A charting Bean (charts data).
- A random number generator Bean.
- Animation Bean (plays animation).
- Timer Bean (produces events).

These Beans could be **connected** in the following way:

- When the timer Bean triggers an event, the random number generator Bean produces a new number.
- This new number is sent to the animation to control its speed.
- The charting Bean records the speed of the animation over time.

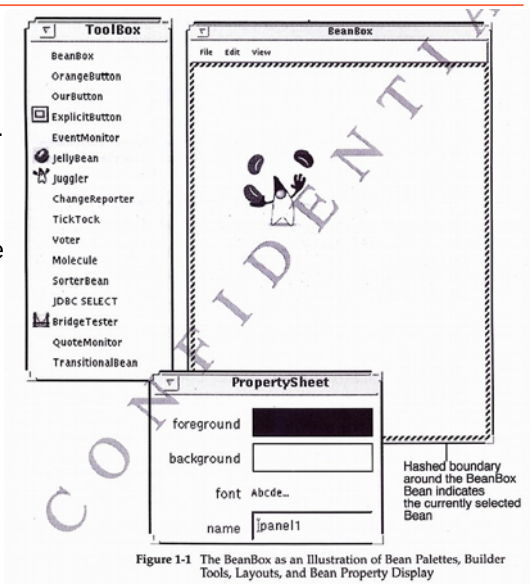


Figure 1-1 The BeanBox as an Illustration of Bean Palettes, Builder Tools, Layouts, and Bean Property Display

© Softwaresysteme. Alle Rechte vorbehalten.

Classification of Beans

Components can cover a wide range of functionality.

Examples:

- **Visual components**:
  - Buttons, sliders, GUI controls.
  - Database viewers, stock data feeds.
- In-house customizable mini-**applications**
  - Word processors, spreadsheets.
- Invisible **server beans**

JavaBeans Technologies

Key technologies explained subsequently:

- Bean events and event handling
- Bean properties
- Introspection
- Customization
- Persistence
- Distribution

© Softwaresysteme. Alle Rechte vorbehalten.

Events in JavaBeans

Beans are interconnected via **events**.

The event model is adopted from the **JDK 1.1 AWT event model**: Observer pattern.

Goals are:

- decoupling of Beans
- ease of connecting methods to events
- strong typing

When a bean fires an event...

- it invokes a named, typed method...
- on a named, typed interface (listener).

Bean must allow registration of listeners.

Bean Properties

Bean property:

- A named attribute of a Bean that can affect its behavior or its appearance.

Types of Bean Properties:

- **Simple**
  A single value whose changes are independent of changes in any other property.
- **Bound**
  A property in which a change results in notifying some other Bean.
- **Constrained** (“vetoable”)
  A property in which a change results in validation by another Bean.
  The change may be rejected.
- **Indexed**
  A range of values instead of a single value is supported.

Accessible via getter/setter methods.

Beans can be interconnected using properties and methods.

Introspection

Problem:

- When you want to include a Bean in your application, you need documentation.
- This documentation should be platform independent and machine-readable (so that a builder tool can use it).

Possible solutions:

- Solution 1: Define a format for a structured text file as a component descriptor file.
- Solution 2: Analyze the Bean's properties, events, and methods. → Introspection

JavaBeans uses solution 2.

The process of finding out about a Bean is called Introspection.

Information on the Bean is deduced from it, and developers can also specify it explicitly.

Advantage: Consistency, avoidance of ad-hoc solutions

Bean Customization

Customization allows you to modify the appearance of a Bean within an application builder to make it suits your needs.

Properties are edited through **property sheets**.
More complex properties like colors and fonts have special pop-up dialogues that coordinate different controls.

Add property sheets for special properties.

- Customization allows you to provide editors.
- Customization also allows you to provide customizers (“wizards”).

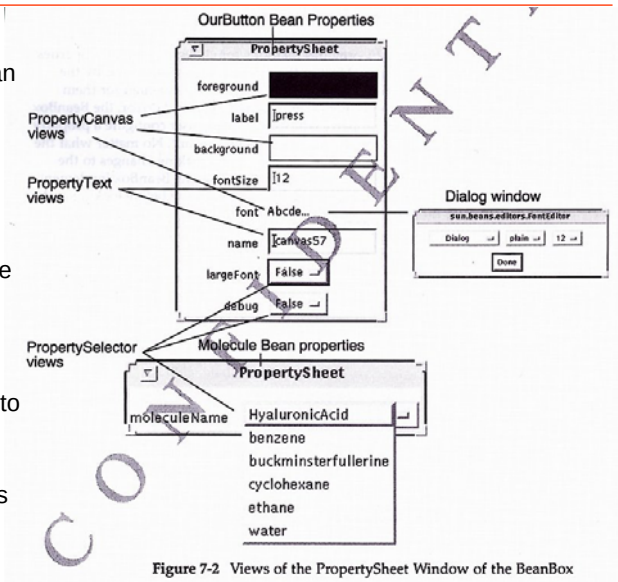


Figure 7-2 Views of the PropertySheet Window of the BeanBox

© Softwaresysteme. Alle Rechte vorbehalten.

Bean Persistence

The properties of a Bean can be changed.

How to store them?

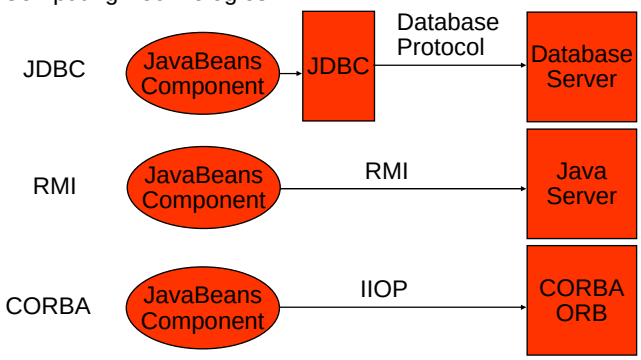
- Make the Bean object persistent.
- Java supports persistence through serialization.

Distributed Beans

Distribution of Beans to support **enterprise services**. Examples:

- Workflow applications: A form is passed around, and is filled out incrementally.
- Application servers: Monitor sensor data.
- Agents: Information plus behavior.

Distributed Computing Technologies:



© Softwaresysteme. Alle Rechte vorbehalten.

Ongoing Beans Development

InfoBus

- For exchanging structured data. http://java.sun.com/products/javabeans/infobus/

JavaBeans Activation Framework

- For using beans as MIME viewers.

http://java.sun.com/products/javabeans/glasgow/jaf.html

Containment and Services Protocol

- For grouping beans into contexts.

http://java.sun.com/j2se/1.3/docs/guide/beans/spec/beancontextTOC.fm.html

Developing JavaBeans Components

Sun offers the **Beans Development Kit** (BDK).

- Build and interconnect JavaBeans components.
- Create well-behaved Bean components.

Technology:

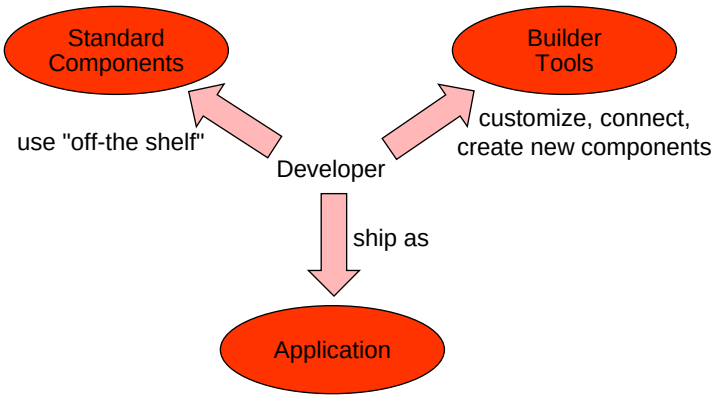
- Event model: Used for interconnection.
- Reflection API: Used for Introspection.
- Object serialization: Used to persistently package beans for reuse.
- Serialized prototypes: Used for pre-customization.
- JAR archives: Used as the standard mechanism for delivering Beans.

Customization:

- Builder tools and human programmers can easily instantiate existing beans.
- Further customization through BeanInfo.

© Softwaresysteme. Alle Rechte vorbehalten.

Beans Development

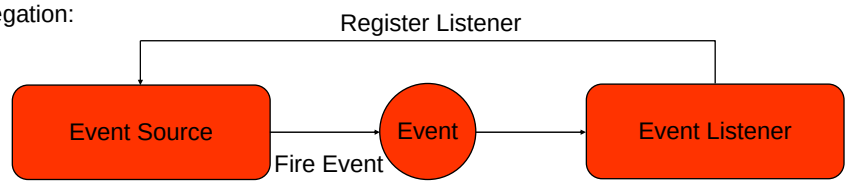


Bean events and event handling (1)

Basis: JDK 1.1 event handling.

- Sources.
- Events.
- Handlers.

Delegation:

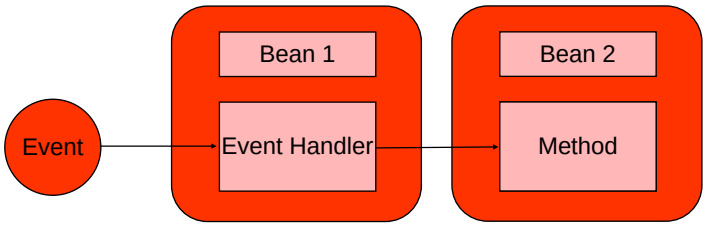


Event sources:

- **Unicast**: Allows only one listener to register for events.
- **Multicast**: Allows multiple listeners to register for events.

Bean events and event handling (2)

Beans are interconnected by using the JDK 1.1 event handling.



Bean events and event handling (3)

Implementing the event model for an event class *X*:

X`Listener` interface:

```
public interface XListener extends EventListener {
        void XHandler (XEvent evt) ;
}// interface XListener
```

The according *X*`Event`:

```
public class XEvent extends EventObject {
        // definition for XEvents
}// class XEvent
```

Event source:
Defines listener registration methods,

```
public void addXListener (XListener l) ;
public void removeXListener (XListener l) ;
```

fires *X*`Event`, notifies listeners of *X*`Event`.

Event listener:
Implements the *X*`Listener` interface, adds itself as a listener of *X*`Events` using

```
source.addXListener () ;
```

Bean Properties

Properties can be read-only, write-only, or read-write, according to the accessor methods.

Accessor Methods:

- The name of the Bean property and the names of the property accessor methods are dependent on each other.
  For a read-write property *prop* of type *PropType* (mind capitalization!):

```
public PropType getProp () ;
public void setProp (PropType prop) ;
```

Exceptions:

- Boolean properties have `isProp ()` and `setProp ()`.
- Indexed properties have

```
public PropType getProp (int index) ;
public void setProp (int index, PropType prop) ;
public PropType [] getProp () ;
public void setProp (PropType [] props) ;
```


Implementing Bean Properties (1)

Example: Bean `ColoredBean` with a simple property `color` (!).

```
import java.awt.Color ;

public class ColoredBean {

    private Color myColor = Color.black ;

    public Color getColor () {
        return myColor ;
    }// getColor

    public void setColor (Color newColor) {
        myColor = newColor ;
    }// setColor

}// ColoredBean
```

Implementing Bean Properties (2)

Defining **bound properties**:

- Accessor methods are defined the same way as for simple properties.
- Additionally the Bean has to be a source of `PropertyChangeEvents` (and thus notify `PropertyChangeListeners` by calling their `propertyChange` method).

Defining **constrained properties**:

- The set method is written as follows:
  - Save the old value.
  - Notify registered `VetoableChangeListeners`.
  - If no listeners vetoes, set the property to the new value, else throw a `PropertyVetoException`.
- When changing the value of a constrained property, the source Bean is responsible for catching exceptions, eventually reverting the old value, and then notifying all listeners of the reversion.

Introspection

Prerequisites for Introspection:

- Reflection.
- Naming convention for properties, events, and methods (the JavaBeans specification calls this “Design Patterns”).

Introspection means...

- reading the explicit information from a Bean, and
- applying reflection according to the naming conventions to complete it.

Explicit information:

- Developers can give specific information about their Bean by supplying an object whose class implements `java.beans.BeanInfo`.
- It returns information when `getBeanInfo()` of `java.beans.Introspector` is invoked.

Reflection:

- Java’s reflection ability allows to inspect objects.

© Softwaresysteme. Alle Rechte vorbehalten.

Uses for BeanInfo

The Introspector can analyze a Bean.

Why additionally provide BeanInfo?

- Limit a long list of properties or events to a few important ones.
- Provide a GIF image as an icon for the builder's component palette.
- Add descriptive, human readable, and possibly *localized* names for properties.
- Make properties "hidden" or "expert" to accommodate different models of development.
- Assist the developer in changing properties.

Bean Customization

Pre-existing property editors (in `java.beans.PropertyEditorManager`):

- BoolEditor
- ByteEditor
- ColorEditor
- DoubleEditor
- FloatEditor
- FontEditor
- IntEditor
- LongEditor
- NumberEditor
- StringEditor

Programmer-defined property editors:

- Implement the `PropertyEditor` interface.
- Define `setValue(Object o)` in a way that makes a *copy* of the object.
- Provide a null argument constructor.
- Support addition and removal of `PropertyChangeListeners`.
- Name your editor *propType*`Editor`.

Implementing a `Customizer` is slightly more complicated.

© Softwaresysteme. Alle Rechte vorbehalten.

Bean Persistence

Java supports persistence through serialization.

Serialization:

- To make objects serializable, their class must implement the `Serializable` interface.
- Serializable objects can be written to/read from a stream.
  Example: Store to/read from a file.
- Serialization includes all fields of an object. Fields that must be excluded have to be marked `transient` or `static`. This is important for
  - References to other Beans.
  - Streams.
  - Threads.

JavaBeans Summary

JavaBeans is the client-side component architecture of Java.

It keeps Java's platform independence, but integrates into native models.

The technologies JavaBeans builds upon are existing Java features:

- Object orientation.
- Reflection.
- Serialization.

Beans can be created with small effort, existing programs are often already programmed in a "Bean-like" way.