*8th Baltic Region Seminar on Engineering Education*
*Kaunas, Lithuania, 2-4 September 2004*

# PhysBeans - a Toolkit to simplify the Construction of Applets for Physics Simulation

**P. Junglas**

Private University of Applied Sciences
Vechta,Diepholz, Germany

ABSTRACT: Constructing a simulation program that can be used for teaching physical phenomena is a lot of work. The concepts of graphical programming, as known from JavaBeans or Visual Basic, could help here: They reduce the programming task mainly to selecting a number of predefined blocks and connecting them with the mouse. PhysBeans is a library of such building blocks specially designed for the simple construction of physical simulations. It is based on the Java Beans model and contains blocks for standard input and display elements, for physical models and their visualisation and for mathematical functions. To show the benefits of this approach the creation of an example applet for the simulation of an electric dipole is demonstrated. Starting point is a graphical block diagram which shows the used building blocks and the flow of information between them. This diagram can simply be rebuilt in a graphical programming environment, reducing the amount of hand written source code to a minimum. PhysBeans has been used to create applets from many different physical areas. The library and all applets are released as open source.

## INTRODUCTION

That physical simulations using Java applets can be a useful didactic tool, has been proven by a lot of applications (f.i. [1], [2]). Unfortunately the construction of such applets is a huge programming effort and needs a profound knowledge of the Java programming language and its libraries. For this reason large lists of applets are collected in the internet (e.g. [3]) which can be used freely. But every lecturer has her or his own way to explain things and different points to make, which leads to a large number of similar applets all created from scratch.

A way out of this dilemma are the physlets [4], a set of general purpose applets, which each span a certain physics topic. They can be configured using JavaScript to such a wide degree that one can construct very differently looking applications from one basic applet. This freedom has its price: In addition to JavaScript one has to learn the special features of the specific physlet.

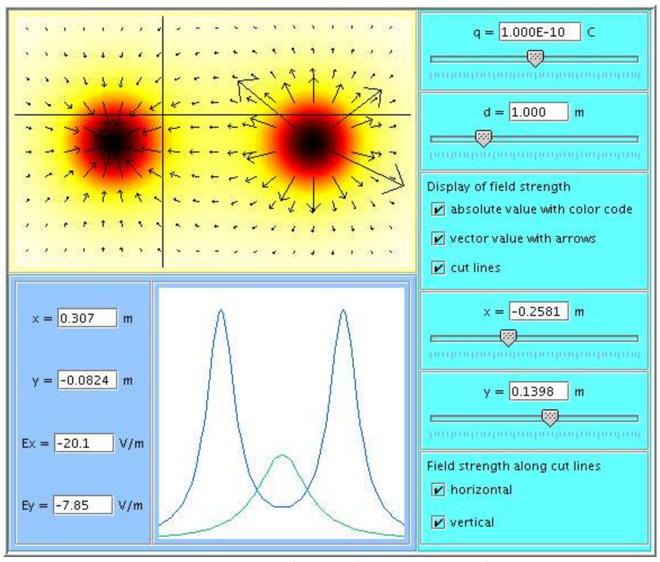
The approach discussed here has a different focus: The PhysBeans library contains a set of building blocks which allow to construct a physical simulation in a graphical way. Little (hopefully none) explicit Java code is needed to create an individual applet that exactly suits the needs of a lecturer. A complete example applet will demonstrate what the PhysBeans library provides and how it is used.

## ELECTRICDIPOLE - AN EXAMPLE APPLET

The applet that will serve as an example in the following is used in a basic course on electromagnetic fields for engineers. It shows the electric field of two opposite point charges.

The electric field is displayed by a grid of arrows, its absolute value is represented using a color table. The value of the charges and their separation can be changed with sliders or by direct numerical input. This gives a clear intuitive understanding of the qualitative properties of the dipole field.

In a next step the students have to find the quantitave behaviour in the far and near field regime. For this purpose the applet allows to measure the electric field strength at an arbitrary point by simply clicking there. Furthermore the absolute value of the field strength along arbitrary horizontal or vertical cut lines is displayed as simple graphs. This allows to use the applet as an "virtual experiment" in the sense of [5],[2].


q = 1.000E-10 C

d = 1.000 m

Display of field strength
[✓] absolute value with color code
[✓] vector value with arrows
[✓] cut lines

x = 0.307 m
y = -0.0824 m
Ex = -20.1 V/m
Ey = -7.85 V/m

x = -0.2581 m

y = 0.1398 m

Field strength along cut lines
[✓] horizontal
[✓] vertical

*Figure 1 Applet "Electric Dipole"*

To program such an applet is quite a lot of work, altogether it uses 47 classes (plus a lot of Java standard classes) consisting of 5049 (non-empty) lines of code, only a minor part of which (13%) is connected with the physical equations or basic mathematics. The rest is mainly used for the user interface and the graphical representation of the data and for general infrastructure purposes like coordinate transformations, color tables or message passing facilities. Almost all of the classes can be (and actually are) reused in other applets. The applet specific part (the "main program") only makes up 6 % of the code. This clearly shows the large potential for a classical class library like [6], which simplifies the task of an applet programmer considerably. But one can make it even simpler using a graphical approach.

## GRAPHICAL PROGRAMMING WITH JAVABEANS

Programming infrastructures like Visual Basic [7] or JavaBeans [8] try to reduce the programming task as much as possible to simple graphical operations. They are especially helpful for the implementation of graphical user interfaces, but can be used in a much broader sense, as we will see in the following.

Writing a program within a JavaBeans environment is reduced mainly to the following steps:

- select the needed components (the Java beans proper) from a palette of predefined beans,
- configure the beans by entering all non-default values for their properties in a property sheet,
- connect the beans with lines, which denote messages going from one bean to another, and define the message content.

Beans have a built-in mechanism to send messages to a list of receivers, whenever their state changes or an external (f.i. user-initiated) event occurs. The different programming environments provide various means, which allow to define messages without writing any explicit code. Nevertheless it is often necessary, and sometimes even simpler, to write some Java lines by hand. The following example will illustrate the way to work with JavaBeans using the free Netbeans environment [9]. It shows clearly what can be done graphically and where hand code still is necessary.

The example program is a simple calculator with two input fields for entering numbers, four buttons with the basic arithmetic operations and an output field displaying the result. The construction of the program proceeds with the following steps:

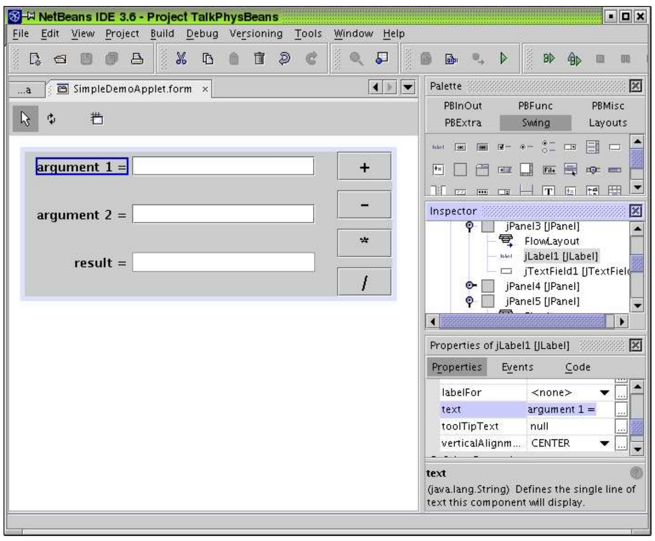
Start by using a simple template for applets. An empty grey rectangle is shown that represents the graphical user interface.

Next add the necessary components from the palette by selecting them and clicking in the rectangle, namely 3 text fields, 3 labels, 4 buttons and 5 panels, simple containers used for nice grouping of the elements.

Now configure the components by providing values for all properties that are different from their defaults. This is done by simply editing the values in the property sheets. Changed properties are

- layout types, border sizes etc. to arrange everything nicely
- text and font of the labels
- size, font and initial text (none) of the text fields
- text and font of the buttons.

This completes the user interface which is displayed in a WYSIWYG style in the NetBeans window.



| | |
|---|---|
| labelFor | <none> |
| text | argument 1 = |
| toolTipText | null |
| verticalAlignm... | CENTER |

text
(java.lang.String) Defines the single line of text this component will display.

*Figure 2 Programming environment NetBeans*

Finally add behaviour by drawing connections and defining the proper messages. Four connections are needed here, one from each button to the output field. The message should say basically:

> "When I (the button) am pressed, you (the output field) have to get the numbers from the two input fields, add (subtract etc.) them and display the result."

This message is a bit too complicated to create it graphically, though this would be possible by inserting an intermediate bean for addition. To see what is possible without any handcode, we first implement the following simpler message:

> "When I (the button) am pressed, you (the output field) have to get the number from the first input field and display it."

After clicking at sender (button) and receiver (output field) with the connection tool, the "connection wizard" pops up and asks for the following information, which can be entered simply by selecting from a list of predefined values:

- reason of the message (button is pressed)
- property to change at the receiver (text)
- origin of text (text property of first text field)

This creates all code that is needed for the message transfer as guarded code (i.e. it cannot be changed manually) and the message content itself:

```
jTextField3.setText(jTextField1.getText());
```

Now the original message can be implemented easily by replacing this code with the following lines:

```
double val1 = Double.parseDouble(jTextField1.getText());
double val2 = Double.parseDouble(jTextField2.getText());
double result = val1 + val2;
jTextField3.setText(result + "");
```

Proceeding in the same way with the other three connections one finishes the applet. The complete program consists of 139 non-empty lines, of which 16 have been handcoded.

## COMPONENTS OF THE PHYSBEANS LIBRARY

Using this graphical programming style is only possible if all the necessary beans exist. Standard environments usually only contain beans from Sun's Swing library [10], which serve as basic building blocks for a graphical user interface. For constructing physical simulations one needs more sophisticated input and output elements and a lot of non-graphical components. This is where the PhysBeans library comes in.

The aim of the PhysBeans project is to provide a set of beans, which make a graphical construction of physical simulation applets possible. Its components fall into five categories:

- Input beans allow the input of information by the user. They are displayed graphically, usually in a special input panel.
  Examples: TextSlider, Timer.
- Output beans display results of measurements or show views of the physical objects.
  Examples: Oscilloscope, VectorTextField, LayeredScreen.
- Physical beans describe physical objects and their behaviour using the specific physical laws. They do not contain any graphical representations.
  Examples: ElectrostaticPointCharges, DrivenPendulum.
- View beans define a specific graphical representation.

Often they are the interface between a physical object and an output element, usually the LayeredScreen.
Examples: ImagePainter, VectorPainter, FunctionPainter.

- Function beans represent mathematical functions. They usually have one or more inputs (arguments) and an output sending the result. They are not displayed graphically.
Examples: FFTFunction, CutFunction.

The task of arranging all elements nicely needs a good understanding of the Java layout mechanism. To free the programmer from this burden, the library additionally contains three applet templates with different predefined arrangements of panels for input and output elements and for the visualisation of the physical objects.

The example beans that are described in more detail in the following, give a good overview of the kind of components that the PhysBeans library provides. Furthermore they are the building blocks of the ElectricDipole applet.

Input beans:

*TextSlider* allows to enter a numerical value using a slider or a text field. Some of its properties are a description and a unit text, the number of significant digits to use for the value and the type of its slider (logarithmic, inverted). It sends a message when a new value is entered.
*SwitchBox* is a collection of options, which can be selected independantly. It cares for the texts and a nice arrangement and sends a message, whenever a box is clicked.

Output beans:

*VectorTextField* displays a complete vector of numerical values, each in a text field of its own. It provides description and unit texts and some layout options.
*LayeredScreen* is a screen with several layers each of which is attached to a different view. Views are drawn in fixed order and can be disabled individually. The LayeredScreen defines a common world coordinate system for all its views, cares for the transformation between screen and world coordinates and has an option to display a crosshair cursor. It listens to mouse clicks and sends messages with the corresponding world coordinates.

Physical beans:

*ElectrostaticPointCharges* is a model of a set of static electric point charges in two dimensions. It computes the electric field strength as a vector field and the absolute value and the potential as scalar fields. The number of charges, their value and their positions can be configured. For conveniance it provides common scale factors for the charges and for the position vectors. It sends a message to notify of general changes and can be asked to send a message with the field strength at a given point.

View beans:

*ImagePainter* draws a color image that represents a scalar field by using a color table to map scalar values to colors. The standard map defines a cold - hot feeling going from blue over white to red.
*VectorPainter* draws arrows at regular grid points representing a vector field. If the length of an arrow is larger than a given limit, the arrow is hidden or a marker may be drawn instead.

*FunctionPainter* draws a vector of two-dimensional points by connecting them with lines of a given color. It is commonly used to create function graphs.
*SimpleFigurePainter* adds simple additional features to an image like a few lines, a rectangle, oval or arc (filled or unfilled) or a string. It's a graphical "Swiss army knife" for all the little things to add. Its properties contain the type of figure, the points defining it and its color.

Function beans:

*CutFunction* is a specialised function that computes a number of field values of a given scalar field along a horizontal or vertical cut line. It outputs a vector of two-dimensional points, each giving the coordinate along the line and the corresponding field value. The position and orientation of the line can be configured, as well as the number of points to compute.

## CONSTRUCTION OF THE EXAMPLE APPLET

The building of a new applet starts with the following preliminary steps:

1. Didactical considerations: What are the basic ideas to learn from it? How can it be used?
2. Design of the user interface: What quantities can be entered, what measurements can be done, what is displayed? This fixes the input and output beans and most of the view beans.
3. Internal function: Which beans are needed to make everything work? Starting with the central physical beans, all necessary mathematical and auxiliary function beans are added.

At the end of this stage one has built up a flow chart that shows all necessary beans and the message flow between them:

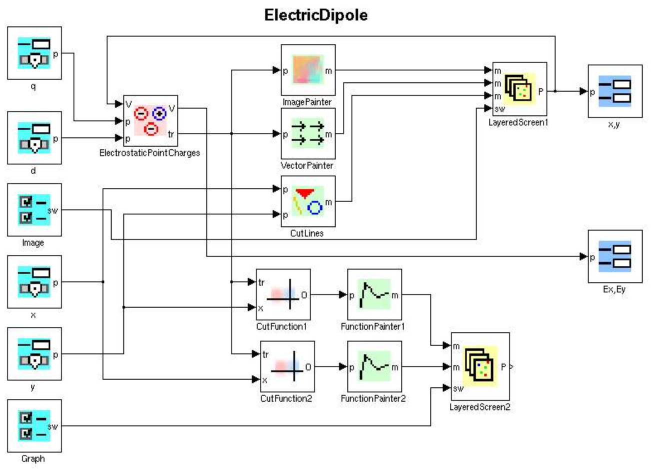


*Figure 3 Flow chart of Electric Dipole applet*

The upper part shows the ElectrostaticPointCharges bean, which defines the two opposite charges, whose value and distance can be set using TextSliders. The ImagePainter and VectorPainter create the corresponding color and arrow images that are displayed on the upper LayeredScreen. An additional element here are the two perpendicular cut lines which are created with a SimpleFigurePainter and whose positions are set with TextSliders. A SwitchBox can be used to choose what will be displayed on the screen.

The lower part contains two CutFunctions which compute the values along the given cut lines. They send their results to FunctionPainters, which create the graphs that are displayed on the second LayeredScreen. The values are recomputed every time the x-, y-sliders send new values or when the physical model is changed. Again a SwitchBox is used to choose which graph to show.

If the user clicks on the upper LayeredScreen, the coordinates are displayed by a VectorTextField. Furthermore they are forwarded to the ElectrostaticPointCharges bean, which in turn sends the corresponding field values to another VectorTextField.

The implementation now proceeds in the same way as in the calculator example:

- Start with one of the standard templates.
- Add all visible and invisible components by selecting and clicking in the proper panel. They can easily be rearranged later with drag and drop.
- Configure all beans by defining f.i. the two charges with opposite values, all description texts and ranges of sliders and the coordinates of the screens.
- Define messages for all connections in the flow chart.

As mentioned above the last step is the only one needing any explicit Java code. For example the message from the TextSlider "y" to the SimpleFigurePainter "CutLines" can not be done in a graphical way directly, because the slider sends a simple number (a double), while the SimpleFigurePainter needs two points to define the vertical line. Such a conflict can be resolved manually with the following lines:

```
double val = textSlider3.getValue();
simpleFigurePainter1.setP3(new DVector(-1.0, val));
simpleFigurePainter1.setP4(new DVector(1.0, val));
```

If one wants to avoid any hand code completely, one could use a simple function bean as an interface, which creates a two-dimensional vector from a number by getting the second coordinate as a fixed parameter.

The complete applet program consists of 277 non-empty lines, of which 21 lines are used to define message contents. Out of these only 7 lines are created manually, but this could be made to zero with little additional effort.

## CONCLUSIONS

The above example shows that the concepts of JavaBeans together with a proper set of components like PhysBeans make it possible to create non-trivial applets of high didactical value almost without any explicit programming. The notion of reusable blocks is especially suited for general controls and displays but can be applied as well to create all necessary internal parts of a virtual experiment. To dispense from the need of writing Java code at all, one has to further address the following points:

- The need of interfacing between different blocks can be reduced to a minimum by the overall design of the kind of messages and by adding a few simple arithmetic blocks.
- The number of custom blocks needed in special situations can be reduced by providing general purpose blocks, which can be configured widely.
- Some of the messages could be easily created graphically with better support from the programming environment. An open source program like NetBeans probably is a good starting point for the necessary extensions.

The PhysBeans library has been used for the applets on the supplemental CD-ROM of [11] and for a course on nonlinear and chaotic systems. It is the basis of a forthcoming multimedia-enhanced course on vibration theory. Existing applets are mainly from linear and nonlinear oscillations, but applets for electrodynamics, optics, thermodynamics and nuclear physics prove its wide applicability. They all can be downloaded from [12].

The future development will concentrate mainly on two points:

- building of new beans (f.i. with 3d graphics) and new applets,
- redesign of the underlying code to make use of the recent OpenSourcePhysics library [6] that contains many classes, which help to create new beans considerably.

The PhysBeans project is still mainly at the beginning. To make it more useful and to proceed faster, the library and all applets are released as open source. It could be a starting point to make physics educators, who want to use applets, more productive by largely simplifying the programming tasks, so that one can concentrate on the design of didactically useful, easily adaptable physical simulation programs.

## REFERENCES

1. Java applets of Department of Didactics, Physics Institute, University Erlangen. http://www.didaktik.physik.uni-erlangen.de/download/applets.htm
2. Junglas, P., *Using applets for physics education: a case study of a course in non-linear systems and chaos*, Proc. 7th Baltic Region Seminar on Engng. Educ. (2003), 61-64.
3. physics web, Interactive Experiments. http://physicsweb.org/resources/Education/Interactive_experiments/
4. Christian, W., Belloni, M., *Physlets - Teaching Physics with Interactive Curricular Material.* Upper Saddle River: Prentice Hall (2001).
5. Junglas, P., *Einsatz von Applets in der Physik-Ausbildung – Fallstudie Nichtlineare Systeme und Chaos*, Global J. of Engng. Educ. 7, 3 (2003), 337-347.
6. *Open Source Physics*, http://www.opensourcephysics.org/
7. *Microsoft Visual Basic 6.0 Reference Library*. Redmond: Microsoft Press (1998).
8. Englander, R., *Developing Java Beans*. Farnham: O'Reilly (2002).
9. Boudreau, T., Glick, J., Greene, S., *NetBeans*. Sevastopol: O'Reilly & Associates (2002).
10. Walrath, K., Campione, M., Huml, A., *The JFC Swing Tutorial*. Boston: Addison-Wesley Professional (2004).
11. Stöcker, H., *Taschenbuch der Physik mit CD-ROM.* Frankfurt am Main: Harri Deutsch (2004).
12. Junglas, P., *PhysBeans homepage*. http://www.peter-junglas.de/fh/physbeans/index.html