

CHAPTER 3

Working with Physlets

“*Physlets*” are JAVA based Physics-Applets developed by Wolfgang Christian and Mario Belloni at Davidson College¹. The JAVA classes that they have developed make it straightforward to construct interactive simulations, and allow researchers to explore different aspects of a model in a dynamic way. From the student’s perspective, when a successful model has been developed (and understood!), a web-page that summarizes the successes and allows visitors to “test” the theory may be made. To accomplish this one must first understand the basic functionality of the different classes provided, and must learn a little bit about scripting. The scripting language used is JavaScript. Much of the material covered here has been adapted from a recent book written by Christian and Belloni[?].

3.1 Animator Basics

In order to understand how to use the animator class, it is important to know that animator will be used to provide an interactive simulation of some physical system. Ideally the simulation will convey the essential character of the underlying physical principles that govern the system. To accomplish this, a typical web page will contain a *script-* to tell objects how to move, an *applet-* to contain the objects and, a *form-* to gather information from visitors . The skeleton of such a page would look like this:

```
<HTML>
<HEAD>
  ...
  <SCRIPT> ... </SCRIPT>
</HEAD>
<BODY>
  <APPLET> ... </APPLET>
  ...
  <FORM> ... </FORM>
</BODY>
</HTML>
```

¹Visit their department’s page at <http://www.davidson.edu>

While there are many new (html tag) components to consider, perhaps the most important aspect for a page author to focus on at this point is that the overall logic is straightforward and that the implementation will be accomplished in an object-oriented way with the JAVA class definitions.

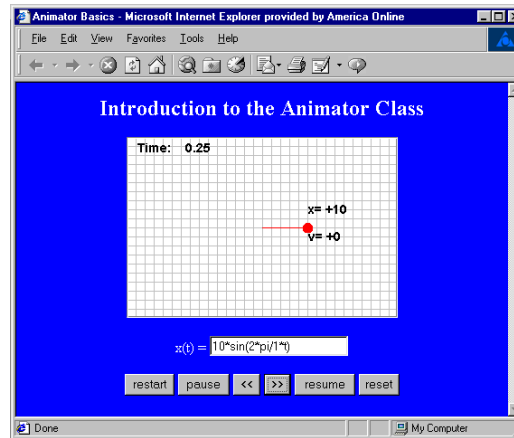


Figure 3.1: The animator class allows page authors to script the motion of objects in an interactive way.

As a rudimentary example of how a simulation may be constructed using the animator class, consider the applet shown in figure 3.1. From the point of view of the author, the most important thing to observe about the page shown in figure 3.1 is that the ball is supposed to move on the trajectory given by, $x(t) = 10 * \sin(2 * \pi / 1 * t)$. This means that at a time of $t = 0.25$, which corresponds to one-quarter of the period, the displacement of the object is a maximum, and the velocity is identically zero! The applet shown this behavior in a succinct way, and the form-button-controls make it easy for a visitor of the page to step through the motion. The question of course is, “how is all of this done?”

3.1.1 Scripting with the Animator Class

As coverage of the elements of the JavaScript language is beyond the scope of this book, the essential piece of information that a reader needs to absorb is that JavaScript, like JAVA and C++, is an object-oriented language. Because of this, the “dot operator” notation is employed and many statements take the form,

```
name=object.class.method;
```

That is, “name” is an *instance* of a “method” defined within a “class” that is contained in an “object.”

As a concrete example of how a method may be *instantiated*, consider the following JavaScript code fragment:

```
ball=document.animator.addObject("circle","r=5");
```

Here, `ball` is an *instance* of the `addObject` method, defined within the `animator` class, that is contained in a `document` object.

The complete script needed to “run” the simulation shown in figure 3.1 is as follows:

```

<SCRIPT language="JavaScript">
function anintro(xtStr)
{
// initialize the applet
  document.Animator.setAutoRefresh(false);
  document.Animator.setDefault();
  document.Animator.setPixPerUnit(5);
  document.Animator.setGridUnit(2);
  document.Animator.shiftPixOrigin(0,0);
// create an object and color it
  ball=document.Animator.addObject("circle","r=5");
  document.Animator.setRGB(ball, 255,0,0);
// define and set the trajectory and trail
  xt=xtStr; yt="0";
  document.Animator.setTrajectory(ball,xt,yt);
  nf=200;
  document.Animator.setTrail(ball,nf);
// define coordinate and velocity and make them "slaves" to object
  xttext=document.Animator.addObject("text","text=x,calc="+xt);
  document.Animator.setDisplayOffset(xttext,0,+15);
  document.Animator.setAnimationSlave(ball,xttext);
  vttext=document.Animator.addObject("text","text=v,calc=vx");
  document.Animator.setDisplayOffset(vttext,0,-15);
  document.Animator.setAnimationSlave(ball,vttext);
// set the length of the animation and advance the animation
  document.Animator.setOneShot(0,12,"Animation Finished");
  document.Animator.setAutoRefresh(true);
  document.Animator.forward();
}
</SCRIPT>

```

This script defines a single function², `anintro`, which requires one argument, `xStr`. The majority of this script is not absolutely essential, but has been provided to illustrate how many of the default properties of the `Animator` class may be changed. For example, the three statements,

```

document.Animator.setPixPerUnit(5);
document.Animator.setGridUnit(2);
document.Animator.shiftPixOrigin(0,0);

```

show how three methods may be used to change the fundamental scale and/or origin of the `Animator` grid. To remove the grid altogether one can use `setGridUnit(0)`.

Perhaps the most significant thing to notice from a programming point of view, is that once a method is instantiated, it may become an instance of another method. I. e., the code fragment,

```

ball=document.Animator.addObject("circle","r=5");
document.Animator.setRGB(ball, 255,0,0);

```

takes `ball`, which is an instance of the `addObject` method, and “sends” it to the `setRGB` method, which colors it red. The convention for the `setRGB` method is that the number triplet, 255, 0, 0 yields red, 0, 255, 0 yields green, and 0, 0, 255 yields blue; other colors may be obtained by choosing a triplet such that the sum of all three numbers of the triplet is 255. While these kinds of constructions are very common in object-oriented programming, **it is important to realize that the JavaScript language is C_aS^e sensitive!**

All that is needed to make an object move along a specific trajectory, is to specify the parametric (in time) equations of the x - and y -coordinates. That is, the $x(t)$ and $y(t)$ functions describing the motion must be known. This may be accomplished using the `setTrajectory` method. E. g.,

²In a manner similar to the C programming language, a function is defined using the general syntax, `function(args){statements}`.

```
xt=xtStr; yt="0";
document animator.setTrajectory(ball,xt,yt);
```

In the present case motion in only one dimension is being scripted. Hence, while the x -coordinate is assigned the value of the input string, the y -component is set to zero.³ One can also use the `setTrail` method to show the path that the object takes. It requires the object name and a number of frames. The time over which the trail will endure depends on the time-step used. Once the number of frames has been reached, the trail begins to recede at the same rate. In the present case two-hundred frames were specified (`nf=200;`); with a time-step of one-tenth (`dt=0.01`), and with the frames-per-second set to one-hundred (`FPS=100`), the trail will begin to recede after two seconds. This behavior is much easier to verify when scripting motion in two dimensions.

Animation Slaves

One of the more useful methods available within the animator class is the `setAnimationSlave` method. It can be used to create compound objects of to attach text and/or a calculation to an object. In the introductory script this method used to “slave” the coordinate and velocity values to the ball. To construct the coordinate text and value, one has to use the `addObject` method to create the text. Once this is accomplished the text can be offset by a specific amount with the `setDisplayOffset` method, so that it will not obscure the object that it is slaved to. I. e.,

```
xtext=document animator.addObject("text","text=x,calc="+xt);
document animator.setDisplayOffset(xtext,0,+15);
document animator.setAnimationSlave(ball,xtext);
```

In a similar manner, the velocity text and value was constructed.

3.1.2 The Applet Tag

In order to place an applet into an html document, an author must use the `<APPLET>` tag. The general syntax for the applet tag is as follows:

```
<APPLET attributes> parameters </APPLET>
```

As an example of how the applet tag may be used, consider that the following code was used to place the applet in figure 3.1.

```
<APPLET codebase="./classes/"
archive="Animator4_.jar,STools4.jar"
code="animator4.Animator.class"
id="animator" name="workspace"
align="center" width="300" height="200">
<!-- set FPS*dt=1 for true time -->
<PARAM name="FPS" value="100">
<PARAM name="dt" value="0.01">
<!-- disable built-in controls -->
<PARAM name="showControls" value="false">
</APPLET>
```

The essential attributes here are the `codebase`, `archive`, and `code` attributes. These indicate respectively, the location of the code⁴, the “.jar” (Java Archive) files needed in that directory, and the class definitions that will be used.

While the `id` attribute is required to identify the class object, the `name` attribute is not. If a name attribute is chosen, then the applet itself may be referenced. The remaining `align`, `width`, and `height` attributes are not necessary- as the default settings may be used.

³By default the input, `xtStr` is of type *string*. Therefore no quotes are needed for the ‘`xt`’ assignment.

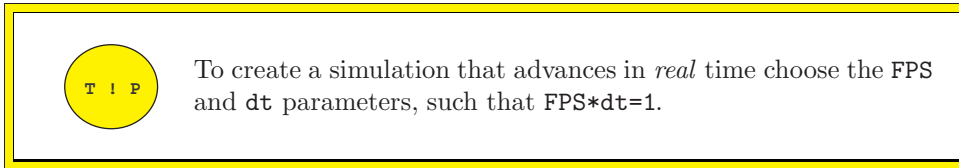
⁴Note that most often the `codebase` attribute is specified using a relative pathname.

Quite often the height and width attributes will be changed so that the applet will take up a specified amount of page, real estate.

In addition to the specifying attributes, page authors may specify different parameters for the animator-class applets. The most important of these are the frames-per-second (FPS) and time-step (dt) parameters. These were set with the following `param` tags:

```
<!-- set FPS*dt=1 for true time -->
<PARAM name="FPS" value="100">
<PARAM name="dt" value="0.01">
```

As indicated, if the FPS and dt parameters are chosen to yield a product of one, then the elapsed time will be *real* time. Other *local* times may be chosen to highlight a particular aspect of a simulation.



3.2 HTML Forms

In order to obtain information from a user, a page author will most often construct a form to “capture” the data. The content of an `html` form is contained within the opening and closing form tags (I.e. , `<FORM>...</FORM>`) and most often contains (at least) a name attribute. The types of input available include: `text`, `password`, `checkbox`, `radio`, `submit`, `reset`, `button`, `hidden` and `image`.⁵ The general syntax for the input controls is as follows:

```
<INPUT type='type' name='name' value='value'>
```

⁵Note that these are available via WinEdt’s drop-down menus.


```
<INPUT type="text" name="xt" value="10*sin(2*pi/1*t)">
```

This means that the string entered within this text input can be referenced using the document object model (DOM) convention⁷ I. e.,

```
document.dataForm.xt.value
```

is identical to the value of the string, and therefore may act as a value for another `method` or may be “passed” to a `function`. More generally, the DOM convention takes the form, `object.class.method.value`, hence one may infer that an HTML form element is a class within a document object, and that the input tag is a method in the form class.

To see how the values are passed, notice that in addition to the `type`, `name` and `value` attributes, many HTML elements also accept the `onclick` attribute. As its’ name suggests, the `onclick` attribute tells the language interpreter what to do when an element is clicked. In many cases more than one action may be desired and as illustrated in the code for the “reset” button in the above form, multiple commands are separated with a semi-colon. I. e.,

```
<INPUT type="button"
      value="reset"
      onclick="document animator.reset();
              document.dataForm.xt.value='10*sin(2*pi*t)';
              JavaScript:anintro(document.dataForm.xt.value)">
```

As always, the spacing is not necessary, but can be very helpful when trying to debug a page!

QUIZ

Modify the contents of the animator introduction page so that a visitor to the page can enter parametric equations for both x - and y -coordinates. To accomplish this you will have to modify the `anintro` function to accept two strings and modify the form to get these values from visitors. Be careful to give some thought to the placement of any new form elements, and provide a brief paragraph telling visitors what your simulation can do. Print out a hard copy of your HTML page and publish your first “Physlet” on your local webserver.

⁷See [5] chapter 9 for more on the DOM.

3.3 Applet Interfacing

In most cases a computer simulation of a physical process should include a graphical representation of the relevant quantities (i. e., position, velocity, etc.) as well as the actual simulation. In order to accomplish this two or more applets will need to be *interfaced* so that information from one applet may be passed to another. To see how this may be accomplished, consider the HTML page shown in figure 3.3 shown below:

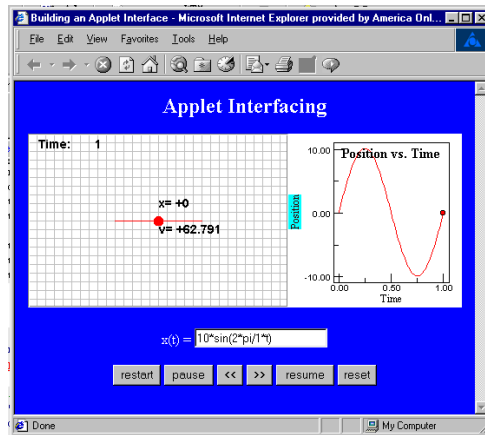


Figure 3.3: Creating an applet interface can greatly enhance the amount of information conveyed by a simulation.

Here the simulation shown in figure 3.1 has been improved upon by adding a second applet to display the position as a function of time.

These graphical abilities are provided for in the `datagraph` class, which includes methods for adding titles and labels to the graph, and (when used in conjunction with the `animator` class) for making data connections with “objects” from other applets. To define the plot and establish the link between the two applets shown in figure 3.3, the following code was added to the `anintro` function script discussed above.

```
// establish a data connection and plot the position
document.dataGraph.setAutoscaleX(true);
document.dataGraph.setAutoscaleY(true);
document.dataGraph.clearSeries(1);
document.dataGraph.setLastPointMarker(1,true);
document.dataGraph.setSeriesStyle(1,true,0);
document.dataGraph.setSeriesRGB(1,255,0,0);
document.dataGraph.setLabelY("Position");
document.dataGraph.setLabelX("Time");
document.dataGraph.setTitle("Position vs. Time");
data=document.dataGraph.getGraphID();
document.animator.deleteDataConnections();
document.animator.makeDataConnection(ball,data,1,"t","x");
```

Note particularly the case in the class identifier, `dataGraph`.

```
01 <APPLET codebase="./classes/"
02     archive="Animator4_.jar,STools4.jar,DataGraph4_.jar"
03     code="animator4.Animator.class"
04     id="animator" name="workspace">
05     align="center" width="300" height="200"
06 <!-- set FPS*dt=1 for true time -->
07 <PARAM name="FPS" value="100">
08 <PARAM name="dt" value="0.01">
09 <!-- disable built-in controls -->
10 <PARAM name="showControls" value="false">
11 </APPLET>
```

(a)

```

01 <APPLET codebase="./classes/"
02     archive="Animator4_.jar,STools4.jar,DataGraph4_.jar"
03     code="dataGraph.DataGraph.class"
04     id="dataGraph" name="graphspace"
05     align="center" width="200" height="200">
06 <PARAM name="Function">
07 <!-- set min/max with name="XMin",value="-10" etc. -->
08 <PARAM name="AutoScaleX" value="true">
09 <PARAM name="AutoScaleY" value="true">
10 <PARAM name="ShowControls" value="false">
11 </APPLET>

```

(b)

Figure 3.4: In order to interface two or more applets it is very important to make sure that the archive attributes of both applets is exactly the same. For example, line 02 in the above applet tags allow a page author to place (a) the simulation, and (b) the graphical representation and to interface them.

By far the most important thing to realize about applet interfacing is that regardless of how or where the applets are placed, **the archive attributes of interfaced applets must be identical**. The code required to place both of the applets in figure 3.3 is shown in figure 3.4 above. The complete code for the page is given in section 3.4

QUIZ

Create an applet interface that allows a user to see a plot of the radius of an object's motion. To accomplish this modify the page used in the last quiz and note that the radius would be given by $r(t) = \sqrt{x(t)^2 + y(t)^2}$. This could be scripted with, `rStr="sqrt(x*x+y*y)";` which would be used as the fifth argument to the `makeDataConnection` method.