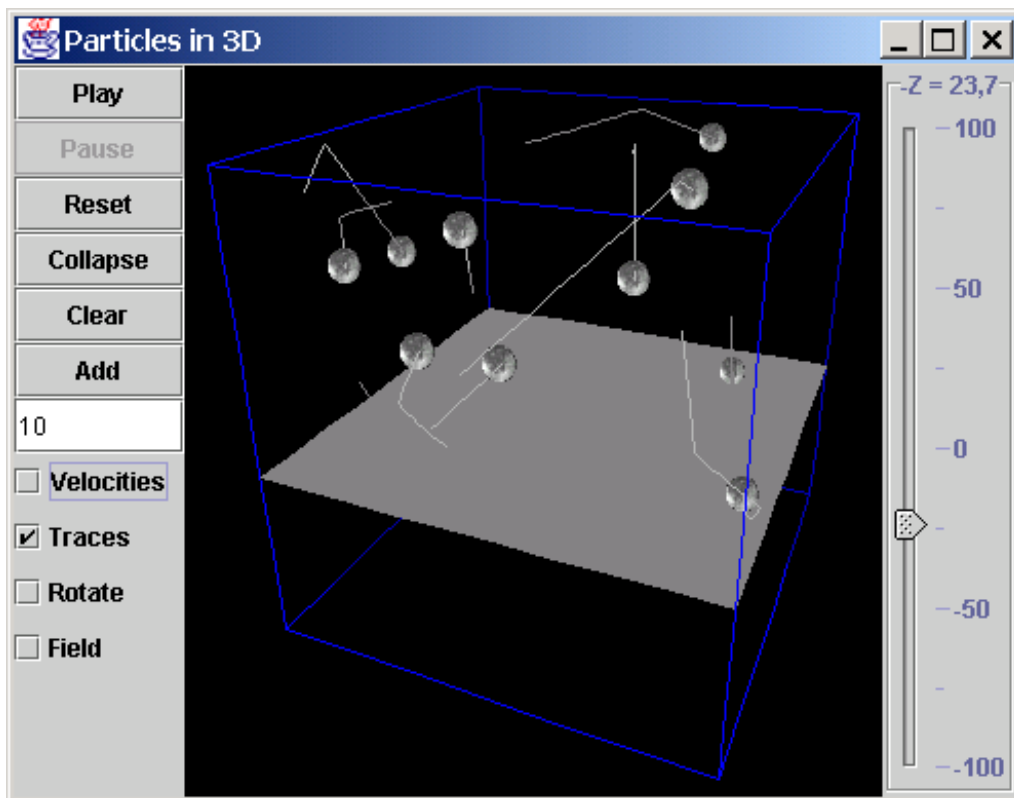




Easy Java Simulations

Appendices for the manual for version 3.1



Francisco Esquembre
Universidad de Murcia

Ejs uses *Open Source Physics* tools
by **Wolfgang Christian**

August 2002
<http://fem.um.es/Ejs>

Contents

A. Programming algorithms in Java.....	3
Declaration of variables	4
Operators	4
Sentences and Expressions.....	6
Bifurcations.....	6
Loops (while, do and for)	7
Special sentences.....	8
Library methods	9
B. Some useful Java classes	11
How to use Java classes	11
java.lang.Math.....	12
java.awt.Color.....	13
java.awt.Font	14
java.awt.Dimension	15
java.awt.Point	15
java.awt.Rectangle.....	15
java.text.DecimalFormat	15
C. About Html.....	19
The body tag.....	19
Block level elements.....	21
Headings	21
Address	21
Paragraphs	22
Lists	22
Preformatted Text.....	24
Div and Center.....	25
Blockquote.....	26
Form	26
Hr - horizontal rules.....	26
Tables	26
Text level elements.....	30
Font style elements	30
Phrase Elements	31
Form fields	32
Select	32
TextArea.....	32
Special Text level Elements.....	32
The A (anchor) element	32
Img - inline images	33
Applet	35
Font.....	37
Basefont.....	38
Br.....	38

Map.....	39
Character Entities for ISO Latin-1.....	40
<i>D. Reference pages for elements for the view.....</i>	43
Containers.....	45
Frame.....	46
Dialog.....	49
Panel.....	52
SplitPanel.....	54
TabbedPanel.....	56
DrawingPanel.....	58
PlottingPanel.....	61
DrawingPanel3D.....	64
Basic elements.....	67
Button.....	68
Checkbox.....	70
RadioButton.....	72
Slider.....	74
Field (or NumberField).....	77
TextField.....	79
Label.....	81
TextArea.....	83
Bar.....	85
Sound.....	87
Drawables.....	89
Particle.....	90
Arrow.....	92
Image.....	94
Text.....	96
Trace.....	98
Poligon.....	100
LightBulb.....	102
ParticleSystem.....	104
ArrowSet.....	107
TextSet.....	113
TraceSet.....	115
Surface.....	117
VectorField.....	119
Lattice.....	122
CheckerField.....	124
Contour.....	126
SurfacePlot.....	128
Sphere.....	130
Cube.....	133
Cilinder.....	135
Cone.....	138
<i>E. Ejs advanced reference.....</i>	141
Personalizing the list of view elements.....	141
Running Ejs with different sets of options.....	142



A. Programming algorithms in Java

“Do I need to learn Java?” - This is perhaps the most frequently heard question when people consider using **Ejs** to create their own simulations.

The answer to this question is ... yes and no.

When writing the equations of a simulation, we certainly need to express our formulas using algorithms in Java language. Hence we need to know a bit of Java. However, what I would call ‘*learning Java*’ is much more than just this bit.

What I mean is that Java is a very complete and powerful programming language, which comes together with a wide set of utility libraries, and learning it in full takes considerable time. But learning just what we need to program our algorithms in Java is much, much simpler.

If you have a look at a book about Java that you may have at hand, you will see that it has a chapter at the beginning of the book devoted to the declaration of variables, to expressions and sentences. I usually refer to this chapter as ‘*chapter 2*’ because it is usually found at the beginning of the book, after an introductory chapter. After this *chapter 2*, you will find that there are a lot more chapters, those which deal with object orientation, classes and many other nice features...But we only need *chapter 2*!

To summarize, the answer to the above question is that you only need to learn how to express your algorithms in Java language and that this is a reasonably accessible task.

You can of course go and read *chapter 2* of your book. However, I have prepared this appendix just in case you want to take a very quick tour through the basic constructions right now: declaration of variables, sentences and expressions, bifurcations and loops. Please do not consider this appendix as a serious tutorial on Java, but it might help you to get started, specially if you have had previous experience with any (yes, I mean any) other programming language.

Finally, having a look at the examples distributed with Ejs will also help you learn what you need.

Declaration of variables

Even when we have chosen to declare variables using our variables editor, sometimes it is necessary to define a local variable (a variable that is only visible within a give page, or within a block of code). This is done by writing the type, the name and, if desired, the initial value. Examples:

```
int i=0; double z;
```

Local variables can only be used within the block in which they are declared. If they are declared at the beginning of one of our model pages, then they can only be used within this page.

Operators

Java operators are:

Arithmetic. These can be either **binary**: addition (+), subtraction (-), multiplication (*) and division (/) or **unary**: plus (+), minus (-), increment (++) and decrement (--).

The last two are less common and are used to increase or decrease, respectively, by one unit, the variable to which they apply. If they appear in an expression before the variable, then the variable is first increased or decreased, and then used in the expression. If they appear after the variable, then just the opposite, it is first used and then changed.

Java includes a special version of the binary addition operator (+) that can be used with constants or variables of type *String*. It is therefore correct to write the sentences:

```
double x = 1.0;  
String text = "The value of x is " + x;
```

which will produce the desired result.

Assignment. These are the equal (=) operator and its combinations with the binary arithmetic operators (+=, -=, *= and /=). Please notice that the expression $x = x + 1$, is not an equation, but implies giving x the value it had, incremented by one. Combinations always have a similar meaning; for instance, $x += 3$ is equivalent to $x = x + 3$.

Comparison. These are used to compare two expressions. The resulting value is always a boolean: *true* or *false*. They are the greater (>), greater or equal

(>=), smaller (<), smaller or equal (<=), equal (=) (please distinguish it from the assignation operator =) and different (!=) operators.

Logical. These are used to build logical expressions, concatenating logical values (true or false): *and* (&&), *or* (||) and *not* (!). Please do not make the (frequent) mistake of writing *and* as a single &, or *or* as a single |.

Finally, there are some special operators called *bit operators*, which we don't cover here, since it is unlikely that you'll use them in your algorithms.

The precedence of the operators is important (what to do in the expression $x*y/z$, for instance?) and usually coincides with the one used when we write mathematical formulas. If there is any doubt, the use of parentheses is recommended.

It is also important to try not to *mix* variables of different types. By this we mean using variables of different types in the same operation. If we are forced to do so, then we must do a *type casting*, which consists in forcing the change of the type of one variable. When this happens from smaller to bigger type, it is done automatically. In the opposite direction, we must do it explicitly.

Assume, for instance, that i is an integer variable (*int*) and x a variable of type *double*. The sentence $x = i*x;$ will cause no problems and the value of i will be converted to a *double* before performing the calculation. However, the compiler would complain at the assignation $i = x*2.0;$ so we must explicitly write $i = (int) (x*2.0);$

A frequent source of errors is to forget that the computer always tries to make a given computation using the simplest possible type of variables. I will exemplify the danger with an example. If you ask the computer to evaluate

```
double x = 1/2;
```

the result is that x will have zero as value! This is because, since 1 and 2 are integers, the computer makes the computation using integer arithmetic, and in integer arithmetic $\frac{1}{2}$ is zero!. To avoid such problems, the correct expression would be

```
double x = 1.0/2.0;
```

Now, the computer realizes that 1.0 and 2.0 are doubles and does the computation properly.

Sentences and Expressions

An expression is a set of variables joined by operators, and it instructs the computer to execute a given operation or operations.

A sentence consists in an expression followed by a semicolon. We usually write a sentence per line to enhance readability, although we can write more than one sentence in the same line.

If a line includes two consecutive bars (`//`) we assume that the line contains a comment. So the computer ignores the part of the line from the two bars to the end.

Typical simple examples of sentences and comments are:

```
// Comment on the next lines
i = 1;
z = 3.0*(x+y); // We compute here the value of z
```

If we want to include a comment that spans through several lines it is better to write these lines between the symbols `/*` and `*/`, instead of writing two bars at the beginning of each line.

Bifurcations

Bifurcations are used to execute only a given sentence from a group of two or more. The first type of bifurcation is given by the conditional clause *if*. Its structure is

```
if (booleanExpression) expr1;
else expr2;
```

At this point, the computer evaluates the expression and, if true, executes the expression *expr1* and, if false, the expression *expr2*. The second sentence, the *else*, is optional; that is, there might be no alternative to executing *expr1*.

If we want to include more than one sentences within an *if-else* we can group them to form a block of code, delimited by the symbols *start of block* (`{`) and *end of block* (`}`). As in

```
if (booleanExpression) {
    expr1;
    expr2;
}
else expr3;
```

(notice that we have indented, to enhance readability, the lines within the block).

The second type of bifurcation is given by the *switch* construction. It is used to build a comparison of the same non-boolean expression with several values. For instance:

```
switch (expression) {
  case value1 : expr1; break;
  case value2 : expr2; break;
  default:     expr3; break;
}
```

where each *case* sentence corresponds to a different possible value of the expression. If the expression takes none of the provided values, then an optional *default* sentence allows executing a sentence.

There may be more than one sentence within each case. The last of these sentences must always be a *break*, indicating the end of the case. If any of these breaks is omitted, then when the case sentence is executed, it also executes the next ones, until a break or the end of the switch construction is found.

Last example can also be written as a sequence of nested *if-else* as follows (though the appearance is not so elegant):

```
if (expression==value1) expr1;
else if (expression==value2) expr2;
else expr3;
```

Finally, there is also a special operator (very seldom used) called *conditional* which, in a sense, can be considered to produce a bifurcation. It has the format

```
booleanExpression ? res1 : res2
```

It evaluates the boolean expression, returning the value *res1* if it is true, and the value *res2* if it is false.

Loops (while, do and for)

A loop is used to execute a sentence, or block of sentences, several times. We indicate one or more logical conditions and, while they are valid, a block of code, delimited again by the symbols *{* and *}*, is executed (brackets can be omitted if there is only one sentence in the block).

Loops can be *while*, *do-while* and *for* constructions. The formats are the following:

```
while (booleanExpression) {
  expr1;
  ...
}

do {
  expr1;
  ...
}
```

```
    } while (booleanExpression);

    for (initialization; booleanExpression; increment) {
        expr1;
        ...;
    }
```

The first two are similar; the only difference is that in the *while* the boolean expression is evaluated before the block of sentences is executed, while in the *do-while* it is evaluated afterwards (which means that the block is executed at least once).

The *for* construction above is equivalent to

```
    initialization;
    while (booleanExpression) {
        expr1;
        ...
        increment;
    }
```

and is used very frequently for processes that are executed an integer number of times. A classical example (which also uses the feature of Java of declaring local variables almost everywhere) is:

```
    for (int i=0;i<10; i++) {
        expr1; // i is often used in these expressions,
              // for instance as the index of an array
        ...;
    }
```

The result is that the block executes exactly ten times. The initialization and increment sentences can hold more than one expression, which must then be separated by commas. Example:

```
    int j, max=10;
    for (int i=0, j=i+100;i<max; i++, j+=2) {
        expr1;
        ...;
    }
```

Special sentences

break. This sentence, which can be used in the block of a bifurcation or a loop, causes the program to leave the block without executing the lines that follow it.

continue. It is used only in loops and stops the current iteration without executing the lines that follow the sentence. Then, the program evaluates the control boolean expression. If this is still true, the loop is executed again.

In both cases, these sentences are included usually as a result of a check (an *if*, for instance) in the middle of the block.

return. This sentence causes the program to leave the method currently executing. In our case, since methods originate from pages of code, the *return* prevents the rest of the page from being executed.

Library methods

Finally, besides the constructions we have seen, every language has a series of libraries of predefined routines (mathematical functions, graphic libraries, ...) that can be used just by including a call to them in the code, as if they were an expression.

In Java, these routines are called *methods* and the libraries *classes*. There are classes and methods for almost anything one can think of, and the art of being an advanced Java programmer includes a deep knowledge of them.

Although a full coverage of all classes is evidently out of the scope of this manual, the next appendix is devoted to introduce to you some of the classes that you may most surely need when creating your simulations with **Ejs**.

Your knowledge of the basic programming of algorithm in Java would be certainly incomplete if you didn't read about the class *Math* in the next appendix. This class lets you use typical mathematical functions that are bound to need while coding your mathematical expressions and formulas.

The good news is that, although the world of classes is a big one, you can live with just a basic knowledge of some of them. Hence I encourage you to have a look at the next appendix.

This page intentionally left blank

B. Some useful Java classes

How to use Java classes

The first thing we need to describe is how to correctly call java library routines, called *methods*, from your code. Methods do not live on their own, but belong to libraries called *classes*. Classes group both methods and data devoted to perform a certain task. There are classes for virtually everything: for mathematical functions, for describing colors for your interface, for fonts, for file access, for internet connections, ...

This appendix is not a tutorial of Java classes and methods, but rather a cook book where you may find the solution for some of the typical tasks that you may want to do when using **Ejs**.

For this reason, though there are two types of methods, *class* methods and *instance* methods, we will consider mostly class (also called *static*) methods, which are easier to use and that most frequently cover all basic needs.

A call to a static method of any these classes is made by putting together the name of the class and the name of the method plus two parentheses, which enclose the calling arguments, if any. For instance, the correct form of calling the mathematical sine function is:

```
y = java.lang.Math.sin (x);
```

where x and y are variables of type *double*.

The name for a java class is always qualified, as you see above. This means that the class usually belongs to a group of classes which in turn form part of a bigger family. For this reason, the name must be very descriptive (and long!), to help distinguish between classes which may have the same name, but belong to a different group or family.

For instance, in the example above, the *sin* method belongs to the class *Math* which lives in the group *lang* of the family *java*. As a second example, the class that creates and handles colors for the interface is called *java.awt.Color*. The class name is *Color*, the group is *awt* and the family is again *java*.

There is an exception. Although the fully qualified name for the class of mathematical functions is `java.lang.Math`, Java makes an exception with it, and in fact with all the classes of the group *java.lang* (the most frequently used), and allows you to call it simply by its name, in this case *Math*. This is why the example above can also be written correctly as

```
y = Math.sin (x);
```

This is very useful to help keep mathematical expressions shorter. But recall that this is an exception and that only works with classes of the group *java.lang*.

We complete this appendix listing some of the classes that you may want to use while coding your simulation in **Ejs**.

java.lang.Math

(or simply Math)

This is certainly the class that you will most frequently use, because you will need it for your mathematical algorithms.

Recall that the proper way of calling any of its static methods is (for instance for the sine function):

```
y = Math.sin (x);
```

where *x* and *y* are variables of type *double*.

Here is the table of its most popular static methods.

Method	Output value
<code>double abs (double x)</code>	Absolute value of <i>x</i>
<code>double acos (double x)</code>	Arc cosine of <i>x</i> , in the range of 0 through π
<code>double asin (double x)</code>	Arc sine of <i>x</i> , in the range of $-\pi/2$ through $\pi/2$
<code>double atan (double x)</code>	Arc tangent of <i>x</i> , in the range of $-\pi/2$ through $\pi/2$
<code>double ceil (double x)</code>	The smallest integer greater than or equal to <i>x</i>

double cos (double x)	Cosine of x
double exp (double x)	Exponential number e raised to the power of x (e^x)
double floor (double x)	The largest integer number smaller than or equal to x
double log (double x)	Natural logarithm (base e) of x ($\ln x$)
double max (double x, double y)	The greater of x and y
int max (int a, int b)	The greater of the integers a and b
double min (double x, double y)	The smaller of x and y
int min (int a, int b)	The smaller of the integers a and b
double pow (double x, double y)	x to the power of y (x^y)
double random ()	A random number between 0.0 and 1.0, excluding 1.0.
double rint (double x)	The integer number closest to x
long round (double x)	The integer number (given as a long) closest to x
double sin (double x)	Sine of x
double sqrt (double x)	Square root of x
double tan (double x)	Tangent of x
double atan2 (double a, double b)	Converts rectangular coordinates (b, a) to polar (r, theta)

java.awt.Color

This class is used to help you describe a color for an element of the interface. The colors are specified by a RGB (red, green and blue) scheme in a way that every color is specified by giving three integer coordinates from 0 to 255, corresponding to the level of red, green and blue basic color components that mix up to create the full color.

For instance, the numbers 255,0,0 correspond to a pure red, while 0,255,0 corresponds to green, 0,0,255 to blue and 255,255,255 corresponds to white.

Besides this, a color can be given a fourth coordinate which specifies its transparency, also from 0 (fully transparent) to 255 (opaque).

Colors need to be *constructed*. That is, they are created by calling one of the special methods of the class called constructors. Constructors always hold the same name as the class (although there may be more than one if they accept different parameters).

For instance, a new color with coordinates 0,192,255 (a pale blue) would be constructed using the call:

```
myColor = new java.awt.Color (0,192,255);
```

where *myColor* is a variable of type *Object*. If we want a semi transparent version of it, we would call

```
myColor = new java.awt.Color (0,192,255,127);
```

The class *java.awt.Color* has also some predefined colors that you can use directly, without the need to create them. For instance, you could include the following code in your simulation:

```
myColor = java.awt.Color.blue;
```

These predefined colors are *black*, *blue*, *cyan*, *darkGray*, *gray*, *green*, *lightGray*, *magenta*, *orange*, *pink*, *red*, *white* and *yellow*.

Finally, any color can be made darker or brighter using the instance methods (an instance method is a method that belongs to the created color, not to the class) *darker()* and *brighter()*, respectively. For instance, you can make your color darker by issuing

```
myColor = myColor.darker();
```

or can turn it into a bright red by issuing

```
myColor = java.awt.Color.red.brighter();
```

java.awt.Font

This class is used to manage fonts for the texts to be written in the simulation interface. A font is specified by giving its *family name*, its *style* and its *size*.

The *family name* is a string that must be chosen from the list of available fonts in your system. The best way to know which font families are available is to use **Ejs** font editor and see what families it offers you.

The *type* must be one of the system constants *java.awt.Font.PLAIN*, *java.awt.Font.BOLD*, *java.awt.Font.ITALIC*, or a bitwise union of these last two, i.e., *java.awt.Font.BOLD | java.awt.Font.ITALIC*.

Finally the *size* is an integer constant that indicates the point size (in pixels of the font).

Similarly to colors, fonts need to be constructed (see *java.awt.Color* above).

For instance a very common medium-size, bold font is created using the sentence

```
myFont = new java.awt.Font ("Dialog", java.awt.Font.BOLD,12);
```

where *myFont* is a variable of type *Object*.

One can also obtain fonts derived from other fonts. This is done using the instance methods *deriveFont (float size)* and *deriveFont (int type)*. This is very useful if you want to change one of the characteristics of your font without affecting the others.

For instance, to make *myFont* above bigger, you can issue

```
myFont = myFont .deriveFont (16.0f);
```

java.awt.Dimension

This class helps you specify sizes. Its elements (or instances) are created using a call to the constructor of the class

```
java.awt.Dimension (int width, int height);
```

This class could be used to change dynamically the size of a basic element of your view.

java.awt.Point

This class helps you specify position in the screen coordinates. Its elements (or instances) are created using a call to the constructor of the class

```
java.awt.Point (int x, int y);
```

This class could be used to change dynamically the location of a basic element of your view.

java.awt.Rectangle

This class helps you specify a rectangular area in the screen coordinates. Its elements (or instances) are created using a call to the constructor of the class

```
java.awt.Rectangle (int x, int y, int width, int height);
```

java.text.DecimalFormat

java.text.DecimalFormat is a Java class that formats decimal numbers. Describing *DecimalFormat* in full would be too lengthy here, but I can extract a basic information (which in most cases suffices) from its reference page.

A DecimalFormat pattern contains a positive and negative subpattern, for example, "#,##0.00;(#,##0.00)". Each subpattern has a prefix, numeric part, and suffix. The negative subpattern is optional; if absent, then the positive subpattern prefixed with the localized minus sign '-' is used as the negative subpattern. That is, "0.00" alone is equivalent to "0.00;-0.00". If there is an explicit negative subpattern, it serves only to specify the negative prefix and suffix; the number of digits, minimal digits, and other characteristics are all the same as the positive pattern. That means that "#,##0.0#;(#)" produces precisely the same behavior as "#,##0.0#;(#,##0.0#)".

The prefixes, suffixes, and various symbols used for infinity, digits, thousands separators, decimal separators, etc. may be set to arbitrary values, and they will appear properly during formatting. However, care must be taken that the symbols and strings do not conflict, or parsing will be unreliable. For example, either the positive and negative prefixes or the suffixes must be distinct for DecimalFormat to be able to distinguish positive from negative values. (If they are identical, then DecimalFormat will behave as if no negative subpattern was specified.) Another example is that the decimal separator and thousands separator should be distinct characters, or parsing will be impossible.

Illegal patterns, such as "#.#.#" or "#.###,###", will cause DecimalFormat to throw an IllegalArgumentException with a message that describes the problem.

Pattern Syntax

```
pattern    := pos_pattern{';' neg_pattern}
pos_pattern := {prefix}number{suffix}
neg_pattern := {prefix}number{suffix}
number     := integer{'.' fraction} {exponent}
prefix     := '\u0000'..\uFFFD' - special_characters
suffix     := '\u0000'..\uFFFD' - special_characters
integer    := min_int | '#' | '#' integer | '#' '; integer
min_int    := '0' | '0' min_int | '0' '; min_int
fraction   := '0'* '#'*
exponent   := 'E' '0' '0'*
```

Notation:

X* 0 or more instances of X
{ X } 0 or 1 instances of X
X | Y either X or Y
X..Y any character from X up to Y, inclusive
S - T characters in S, except those in T

Practical hint

After all this terrible jargon, here goes a practical suggestion. Use patterns like this one

```
Name = #.00;Name = - #.00
```

if you want to have two decimal points (add more zeroes if you want more) for doubles and a pattern like

```
Name = 0;Name = - 0
```

for integers. In both cases substitute *Name* with the name of the variable which you want to display.

This page intentionally left blank



C. About Html

In order to write nice introductory pages in **Ejs** you need to know the basics of Html language. The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another.

Learning the basics of it is rather an easy task, since writing Html text consist in writing what you want to write, together with some tags here and there that provide some organization and/or different visual appearance to your text.

Surely you can find several good books on the subject, but as I did with Java, I have prepared an appendix on the subject for you.

Instead of writing my own introduction to html, I have taken the text that follows from the original source at <http://www.w3.org/TR/REC-html32.html>. This is not however the complete text, I have edited it to fit my interests.

In particular, since **Ejs** takes care of the head part of the Html text, only tags that apply to the body part of it are included.

Again, please do not take this appendix as a serious tutorial on Html, but rather as a quick revision of Html most important tags.

Note: Theoretically, the html editor included with **Ejs** should accept html text in its version 3.2, with only some minor exceptions (at least this is what the reference for this editor reads). I know that these exceptions include the applet tag (which I found reasonable). However, I haven't been able to find a list of all unsupported html 3.2 features.

The body tag

The key attributes are: BACKGROUND, BGCOLOR, TEXT, LINK, VLINK and ALINK. These can be used to set a repeating background image, plus background and foreground colors for normal text and hypertext links.

Example:

```
<body bgcolor=white text=black link=red vlink=maroon alink=fuchsia>
```

bgcolor

Specifies the background color for the document body. See below for the syntax of color values.

text

Specifies the color used to stroke the document's text. This is generally used when you have changed the background color with the `BGCOLOR` or `BACKGROUND` attributes.

link

Specifies the color used to stroke the text for unvisited hypertext links.

vlink

Specifies the color used to stroke the text for visited hypertext links.

alink


Specifies the highlight color used to stroke the text for hypertext links at the moment the user clicks on the link.

background

Specifies a URL for an image that will be used to tile the document background.

Colors are given in the sRGB color space as hexadecimal numbers (e.g. `COLOR="#C0FFC0"`), or as one of 16 widely understood color names. These colors were originally picked as being the standard 16 colors supported with the Windows VGA palette.

Color names and sRGB values

 Black = "#000000"	 Green = "#008000"
 Silver = "#C0C0C0"	 Lime = "#00FF00"
 Gray = "#808080"	 Olive = "#808000"
 White = "#FFFFFF"	 Yellow = "#FFFF00"
 Maroon = "#800000"	 Navy = "#000080"
 Red = "#FF0000"	 Blue = "#0000FF"
 Purple = "#800080"	 Teal = "#008080"



Fuchsia = "#FF00FF"



Aqua = "#00FFFF"

Block level elements

Most elements that can appear in the document body fall into one of two groups: block level elements which cause paragraph breaks, and text level elements which don't. Common block level elements include H1 to H6 (headers), P (paragraphs) LI (list items), and HR (horizontal rules). Common text level elements include EM, I, B and FONT (character emphasis), A (hypertext links), IMG and APPLET (embedded objects) and BR (line breaks). Note that block elements generally act as containers for text level and other block level elements (excluding headings and address elements), while text level elements can only contain other text level elements.

Headings

H1, H2, H3, H4, H5 and H6 are used for document headings. You always need the start and end tags. H1 elements are more important than H2 elements and so on, so that H6 elements define the least important level of headings. More important headings are generally rendered in a larger font than less important ones. Use the optional ALIGN attribute to set the text alignment within a heading, e.g.

```
<H1 ALIGN=CENTER> ... centered heading ... </H1>
```

The default is left alignment, but this can be overridden by an enclosing DIV or CENTER element.

Address

The ADDRESS element requires start and end tags, and specifies information such as authorship and contact details for the current document. User agents should render the content with paragraph-breaks before and after. Note that the content is restricted to paragraphs, plain text and text-like elements.

Example:

```
<ADDRESS>  
Newsletter editor<BR>  
J.R. Brown<BR>  
8723 Buena Vista, Smallville, CT 01234<BR>  
Tel: +1 (123) 456 7890  
</ADDRESS>
```

Paragraphs

The `P` element is used to markup paragraphs. It is a container and requires a start tag. The end tag is optional as it can always be inferred by the parser. User agents should place paragraph breaks before and after `P` elements. The rendering is user agent dependent, but text is generally wrapped to fit the space available.

Example:

```
<P>This is the first paragraph.  
<P>This is the second paragraph.
```

Paragraphs are usually rendered flush left with a ragged right margin. The `ALIGN` attribute can be used to explicitly specify the horizontal alignment:

```
align=left
```

The paragraph is rendered flush left.

```
align=center
```

The paragraph is centered.

```
align=right
```

The paragraph is rendered flush right.

For example:

```
<p align=center>This is a centered paragraph.  
<p align=right>and this is a flush right paragraph.
```

The default is left alignment, but this can be overridden by an enclosing `DIV` or `CENTER` element.

Lists

List items can contain block and text level items, including nested lists, although headings and address elements are excluded.

UNORDERED LISTS

Unordered lists take the form:

```
<UL>  
  <LI> ... first list item  
  <LI> ... second list item  
  ...  
</UL>
```

The `UL` element is used for unordered lists. Both start and end tags are always needed. The `LI` element is used for individual list items. The end tag for `LI` elements can always be omitted. Note that `LI` elements can contain nested

lists. The `COMPACT` attribute can be used as a hint to the user agent to render lists in a more compact style.

The `TYPE` attribute can be used to set the bullet style on `UL` and `LI` elements. The permitted values are "disc", "square" or "circle". The default generally depends on the level of nesting for lists.

- with `<li type=disc>`
- with `<li type=square>`
- with `<li type=circle>`

ORDERED (I.E. NUMBERED) LISTS

Ordered (i.e. numbered) lists take the form:

```
<OL>
  <LI> ... first list item
  <LI> ... second list item
  ...
</OL>
```

The `OL START` attribute can be used to initialize the sequence number (by default it is initialized to 1). You can set it later on with the `VALUE` attribute on `LI` elements. Both of these attributes expect integer values. You can't indicate that numbering should be continued from a previous list, or to skip missing values without giving an explicit number.

The `COMPACT` attribute can be used as a hint to the user agent to render lists in a more compact style. The `OL TYPE` attribute allows you to set the numbering style for list items:

Type	Numbering style	
1	Arabic numbers	1, 2, 3, ...
a	lower alpha	a, b, c, ...
A	upper alpha	A, B, C, ...
i	lower roman	i, ii, iii, ...
I	upper roman	I, II, III, ...

DEFINITION LISTS

Definition lists take the form:

```
<DL>
  <DT> term name
  <DD> term definition
```

```
...  
</DL>
```

DT elements can only act as containers for text level elements, while DD elements can hold block level elements as well, excluding headings and address elements.

For example:

```
<DL>  
<DT>Term 1<dd>This is the definition of the first term.  
<DT>Term 2<dd>This is the definition of the second term.  
</DL>
```

which could be rendered as:

```
Term 1  
  This is the definition of the first term.  
Term 2  
  This is the definition of the second term.
```

The COMPACT attribute can be used with the DL element as a hint to the user agent to render lists in a more compact style.

DIR AND MENU

These elements have been part of HTML from the early days. They are intended for unordered lists similar to UL elements. User agents are recommended to render DIR elements as multicolumn directory lists, and MENU elements as single column menu lists. In practice, Mosaic and most other user agents have ignored this advice and instead render DIR and MENU in an identical way to UL elements.

Preformatted Text

The PRE element can be used to include preformatted text. User agents render this in a fixed pitch font, preserving spacing associated with white space characters such as space and newline characters. Automatic word-wrap should be disabled within PRE elements.

PRE has the same content model as paragraphs, excluding images and elements that produce changes in font size, e.g. IMG, BIG, SMALL, SUB, SUP and FONT.

A few user agents support the WIDTH attribute. It provides a hint to the user agent of the required width in characters. The user agent can use this to select an appropriate font size or to indent the content appropriately.

Here is an example of a `PRE` element; a verse from Shelley (To a Skylark):

```
<PRE>
  Higher still and higher
    From the earth thou springest
  Like a cloud of fire;
    The blue deep thou wingest,
  And singing still dost soar, and soaring ever singest.
</PRE>
```

which is rendered as:

```
Higher still and higher
  From the earth thou springest
Like a cloud of fire;
  The blue deep thou wingest,
And singing still dost soar, and soaring ever singest.
```

The horizontal tab character (encoded in Unicode, US ASCII and ISO 8859-1 as decimal 9) should be interpreted as the smallest non-zero number of spaces which will leave the number of characters so far on the line as a multiple of 8. Its use is strongly discouraged since it is common practice when editing to set the tab-spacing to other values, leading to misaligned documents.

XMP, LISTING AND PLAINTEXT

These are obsolete tags for preformatted text that predate the introduction of `PRE`. User agents may support these for backwards compatibility. Authors should avoid using them in new documents!

Div and Center

`DIV` elements can be used to structure HTML documents as a hierarchy of divisions. The `ALIGN` attribute can be used to set the default horizontal alignment for elements within the content of the `DIV` element. Its value is restricted to `LEFT`, `CENTER` or `RIGHT`, and is defined in the same way as for the paragraph element `<P>`.

Note that because `DIV` is a block-like element it will terminate an open `P` element. Other than this, user agents are not expected to render paragraph breaks before and after `DIV` elements. `CENTER` is directly equivalent to `DIV` with `ALIGN=CENTER`. Both `DIV` and `CENTER` require start and end tags.

`CENTER` was introduced by Netscape before they added support for the HTML 3.0 `DIV` element. It is retained in HTML 3.2 on account of its widespread deployment.

Blockquote

This is used to enclose block quotations from other works. Both the start and end tags are required. It is often rendered indented, e.g.

They went in single file, running like hounds on a strong scent, and an eager light was in their eyes. Nearly due west the broad swath of the marching Orcs tramped its ugly slot; the sweet grass of Rohan had been bruised and blackened as they passed.

from "The Two Towers" by J.R.R. Tolkien.

Form

Note: Since it is very unlikely that you want to include input forms into your introductions, I have suppressed the information about this tag. Please consult the web page referenced above to learn more about forms.

Hr - horizontal rules

Horizontal rules may be used to indicate a change in topic. In a speech based user agent, the rule could be rendered as a pause.

HR elements are not containers so the end tag is forbidden. The attributes are: ALIGN, NOSHADE, SIZE and WIDTH.

align

This determines whether the rule is placed at the left, center or right of the space between the current left and right margins for `align=left`, `align=center` or `align=right` respectively. By default, the rule is centered.

noshade

This attribute requests the user agent to render the rule in a solid color rather than as the traditional two colour "groove".

size

This can be used to set the height of the rule in pixels.

width

This can be used to set the width of the rule in pixels (e.g. `width=100`) or as the percentage between the current left and right margins (e.g. `width="50%"`). The default is 100%.

Tables

Tables take the general form:

```
<TABLE BORDER=3 CELLSPACING=2 CELLPADDING=2 WIDTH="80%">
<CAPTION> ... table caption ... </CAPTION>
<TR><TD> first cell <TD> second cell
<TR> ...
...
</TABLE>
```

The attributes on `TABLE` are all optional. By default, the table is rendered without a surrounding border. The table is generally sized automatically to fit the contents, but you can also set the table width using the `WIDTH` attribute. `BORDER`, `CELLSPACING` and `CELLPADDING` provide further control over the table's appearance. Captions are rendered at the top or bottom of the table depending on the `ALIGN` attribute.

Each table row is contained in a `TR` element, although the end tag can always be omitted. Table cells are defined by `TD` elements for data and `TH` elements for headers. Like `TR`, these are containers and can be given without trailing end tags. `TH` and `TD` support several attributes: `ALIGN` and `VALIGN` for aligning cell content, `ROWSPAN` and `COLSPAN` for cells which span more than one row or column. A cell can contain a wide variety of other block and text level elements including form fields and other tables.

The `TABLE` element always requires both start and end tags. It supports the following attributes:

align

This takes one of the case insensitive values: `LEFT`, `CENTER` or `RIGHT`. It specifies the horizontal placement of the table relative to the current left and right margins. It defaults to left alignment, but this can be overridden by an enclosing `DIV` or `CENTER` element.

width

In the absence of this attribute the table width is automatically determined from the table contents. You can use the `WIDTH` attribute to set the table width to a fixed value in pixels (e.g. `WIDTH=212`) or as a percentage of the space between the current left and right margins (e.g. `WIDTH="80%"`).

border

This attribute can be used to specify the width of the outer border around the table to a given number of pixels (e.g. `BORDER=4`). The value can be set to zero to suppress the border altogether. In the absence of this attribute the border should be suppressed. Note that some browsers also accept `<TABLE BORDER>` with the same semantics as `BORDER=1`.

cellspacing

In traditional desktop publishing software, adjacent table cells share a common border. This is not the case in HTML. Each cell is given its own border which is separated from the borders around neighboring cells. This separation can be set in pixels using the `CELLSPACING` attribute, (e.g. `CELLSPACING=10`). The same value also determines the separation between the table border and the borders of the outermost cells.

cellpadding

This sets the padding in pixels between the border around each cell and the cell's contents.

The `CAPTION` element has one attribute `ALIGN` which can be either `ALIGN=TOP` or `ALIGN=BOTTOM`. This can be used to force the caption to be placed above the top or below the bottom of the table respectively. Most user agents default to placing the caption above the table. `CAPTION` always requires both start and end tags. Captions are limited to plain text and text-level elements as defined by the `%text` entity. Block level elements are not permitted.

The `TR` or table row element requires a start tag, but the end tag can always be left out. `TR` acts as a container for table cells. It has two attributes:

align

Sets the default horizontal alignment of cell contents. It takes one of the case insensitive values: `LEFT`, `CENTER` or `RIGHT` and plays the same role as the `ALIGN` attribute on paragraph elements.

valign

This can be used to set the default vertical alignment of cell contents within each cell. It takes one of the case insensitive values: `TOP`, `MIDDLE` or `BOTTOM` to position the cell contents at the top, middle or bottom of the cell respectively.

There are two elements for defining table cells. `TH` is used for header cells and `TD` for data cells. This distinction allows user agents to render header and data cells in different fonts, and enables speech based browsers to do a better job. The start tags for `TH` and `TD` are always needed but the end tags can be left out. Table cells can have the following attributes:

nowrap

The presence of this attribute disables automatic word wrap within the contents of this cell (e.g. `<TD NOWRAP>`). This is equivalent to using the ` ` entity for non-breaking spaces within the content of the cell.

rowspan

This takes a positive integer value specifying the number of rows spanned by this cell. It defaults to one.

colspan

This takes a positive integer value specifying the number of columns spanned by this cell. It defaults to one.

align

Specifies the default horizontal alignment of cell contents, and overrides the `ALIGN` attribute on the table row. It takes the same values: `LEFT`, `CENTER` and `RIGHT`. If you don't specify an `ALIGN` attribute value on the cell, the default is left alignment for `<td>` and center alignment for `<th>` although you can override this with an `ALIGN` attribute on the `TR` element.

valign

Specifies the default vertical alignment of cell contents, overriding the `VALIGN` attribute on the table row. It takes the same values: `TOP`, `MIDDLE` and `BOTTOM`. If you don't specify a `VALIGN` attribute value on the cell, the default is middle although you can override this with a `VALIGN` attribute on the `TR` element.

width

Specifies the suggested width for a cell content in pixels excluding the cell padding. This value will normally be used except when it conflicts with the width requirements for other cells in the same column.

height

Specifies the suggested height for a cell content in pixels excluding the cell padding. This value will normally be used except when it conflicts with the height requirements for other cells in the same row.

Tables are commonly rendered in bas-relief, raised up with the outer border as a bevel, and individual cells inset into this raised surface. Borders around

individual cells are only drawn if the cell has explicit content. White space doesn't count for this purpose with the exception of ` `;

The algorithms used to automatically size tables should take into account the minimum and maximum width requirements for each cell. This is used to determine the minimum and maximum width requirements for each column and hence for the table itself.

Cells spanning more than one column contribute to the widths of each of the columns spanned. One approach is to evenly apportion the cell's minimum and maximum width between these columns, another is to weight the apportioning according to the contributions from cells that don't span multiple columns.

For some user agents it may be necessary or desirable to break text lines within words. In such cases a visual indication that this has occurred is advised.

The minimum and maximum width of nested tables contribute to the minimum and maximum width of the cell in which they occur. Once the width requirements are known for the top level table, the column widths for that table can be assigned. This allows the widths of nested tables to be assigned and hence in turn the column widths of such tables. If practical, all columns should be assigned at least their minimum widths. It is suggested that any surplus space is then shared out proportional to the difference between the minimum and maximum width requirements of each column.

Note that pixel values for width and height refer to screen pixels, and should be multiplied by an appropriate factor when rendering to very high resolution devices such as laser printers. For instance if a user agent has a display with 75 pixels per inch and is rendering to a laser printer with 600 dots per inch, then the pixel values given in HTML attributes should be multiplied by a factor of 8.

Text level elements

These don't cause paragraph breaks. Text level elements that define character styles can generally be nested. They can contain other text level elements but not block level elements.

Font style elements

These all require start and end tags, e.g.

This has some `bold text`.

Text level elements must be properly nested - the following is in error:

This has some bold and <I>italic text</I>.

User agents should do their best to respect nested emphasis, e.g.

This has some bold and <I>italic text</I>.

Where the available fonts are restricted or for speech output, alternative means should be used for rendering differences in emphasis.

TT teletype or monospaced text

I italic text style

B bold text style

U underlined text style

STRIKE strike-through text style

BIG places text in a large font

SMALL places text in a small font

SUB places text in subscript style

SUP places text in superscript style

Note: future revisions to HTML may phase out STRIKE in favor of the more concise "S" tag from HTML 3.0.

Phrase Elements

These all require start and end tags, e.g.

This has some emphasized text.

EM basic emphasis typically rendered in an italic font

STRONG strong emphasis typically rendered in a bold font

DFN defining instance of the enclosed term

CODE used for extracts from program code

SAMP used for sample output from programs, and scripts etc.

KBD used for text to be typed by the user

VAR used for variables or arguments to commands

CITE used for citations or references to other sources

Form fields

Select

TextArea

Note: Since it is very unlikely that you want to include input fields (forms, select menus and text areas) into your introductions, I have suppressed the information about these tag. Please consult the web page referenced above to learn more about forms and menus.

Special Text level Elements

The A (anchor) element

Anchors can't be nested and always require start and end tags. They are used to define hypertext links and also to define named locations for use as targets for hypertext links, e.g.

The way to `happiness`.

and also to define named locations for use as targets for hypertext links, e.g.

`<h2>545 Tech Square - Hacker's Paradise</h2>`

name

This should be a string defining unique name for the scope of the current HTML document. `NAME` is used to associate a name with this part of a document for use with URLs that target a named section of a document.

href

Specifies a URL acting as a network address for the linked resource. This could be another HTML document, a PDF file or an image etc.

rel

The forward relationship also known as the "link type". It can be used to determine to how to deal with the linked resource when printing out a collection of linked resources.

rev

This defines a reverse relationship. A link from document A to document B with `REV=relation` expresses the same relationship as a link from B to A with `REL=relation`. `REV=made` is sometimes used to identify the

document author, either the author's email address with a mailto URL, or a link to the author's home page.

title

An advisory title for the linked resource.

img - inline images

Used to insert images. IMG is an empty element and so the end tag is forbidden. Images can be positioned vertically relative to the current textline or floated to the left or right. See BR with the CLEAR attribute for control over textflow. *e.g.*

```
<IMG SRC="canyon.gif" ALT="Grand Canyon">
```

IMG elements support the following attributes:

src

This attribute is required for every IMG element. It specifies a URL for the image resource, for instance a GIF, JPEG or PNG image file.

alt

This is used to provide a text description of the image and is vital for interoperability with speech-based and text only user agents.

align

This specifies how the image is positioned relative to the current textline in which it occurs:

align=top

positions the top of the image with the top of the current text line. User agents vary in how they interpret this. Some only take into account what has occurred on the text line prior to the IMG element and ignore what happens after it.

align=middle

aligns the middle of the image with the baseline for the current textline.

align=bottom

is the default and aligns the bottom of the image with the baseline.

align=left

floats the image to the current left margin, temporarily changing this margin, so that subsequent text is flowed along the image's righthand side. The rendering depends on whether there is any left aligned text or images that appear earlier than the current image in the markup. Such text (but not images) generally

forces left aligned images to wrap to a new line, with the subsequent text continuing on the former line.

align=right

floats the image to the current right margin, temporarily changing this margin, so that subsequent text is flowed along the image's lefthand side. The rendering depends on whether there is any right aligned text or images that appear earlier than the current image in the markup. Such text (but not images) generally forces right aligned images to wrap to a new line, with the subsequent text continuing on the former line.

Note that some browsers introduce spurious spacing with multiple left or right aligned images. As a result authors can't depend on this being the same for browsers from different vendors. See BR for ways to control text flow.

width

Specifies the intended width of the image in pixels. When given together with the height, this allows user agents to reserve screen space for the image before the image data has arrived over the network.

height

Specifies the intended height of the image in pixels. When given together with the width, this allows user agents to reserve screen space for the image before the image data has arrived over the network.

border

When the IMG element appears as part of a hypertext link, the user agent will generally indicate this by drawing a colored border (typically blue) around the image. This attribute can be used to set the width of this border in pixels. Use `border=0` to suppress the border altogether. User agents are recommended to provide additional cues that the image is clickable, e.g. by changing the mouse pointer.

hspace

This can be used to provide white space to the immediate left and right of the image. The HSPACE attribute sets the width of this white space in pixels. By default HSPACE is a small non-zero number.

vspace

This can be used to provide white space above and below the image. The VSPACE attribute sets the height of this white space in pixels. By default VSPACE is a small non-zero number.

usemap

This can be used to give a URL fragment identifier for a client-side image map defined with the MAP element.

ismap

When the IMG element is part of a hypertext link, and the user clicks on the image, the ISMAP attribute causes the location to be passed to the server. This mechanism causes problems for text-only and speech-based user agents. Whenever its possible to do so use the MAP element instead.

Here is an example of how you use ISMAP:

```
<a href="/cgibin/navbar.map"><img src=navbar.gif ismap border=0></a>
```

The location clicked is passed to the server as follows. The user agent derives a new URL from the URL specified by the HREF attribute by appending '?' the x coordinate ',' and the y coordinate of the location in pixels. The link is then followed using the new URL. For instance, if the user clicked at at the location $x=10$, $y=27$ then the derived URL will be: `"/cgibin/navbar.map?10,27"`. It is generally a good idea to suppress the border and use graphical idioms to indicate that the image is clickable.

Note that pixel values refer to screen pixels, and should be multiplied by an appropriate factor when rendering to very high resolution devices such as laser printers. For instance if a user agent has a display with 75 pixels per inch and is rendering to a laser printer with 600 dots per inch, then the pixel values given in HTML attributes should be multiplied by a factor of 8.

Applet

Requires start and end tags. This element is supported by all Java enabled browsers. It allows you to embed a Java applet into HTML documents.

APPLET uses associated PARAM elements to pass parameters to the applet.

Following the PARAM elements, the content of APPLET elements should be used to provide an alternative to the applet for user agents that don't support Java. Java-compatible browsers ignore this extra HTML code. You can use it to show a snapshot of the applet running, with text explaining what the applet does. Other possibilities for this area are a link to a page that is more useful for the Java-ignorant browser, or text that taunts the user for not having a Java-compatible browser.

Here is a simple example of a Java applet:

```
<applet code="Bubbles.class" width=500 height=500>
```

Java applet that draws animated bubbles.

```
</applet>
```

Here is another one using a `PARAM` element:

```
<applet code="AudioItem" width=15 height=15>  
<param name=snd value="Hello.au|Welcome.au">  
Java applet that plays a welcoming sound.  
</applet>
```

codebase = codebaseURL

This optional attribute specifies the base URL of the applet -- the directory or folder that contains the applet's code. If this attribute is not specified, then the document's URL is used.

code = appletFile

This required attribute gives the name of the file that contains the applet's compiled Applet subclass. This file is relative to the base URL of the applet. It cannot be absolute.

alt = alternateText

This optional attribute specifies any text that should be displayed if the browser understands the `APPLET` tag but can't run Java applets.

name = appletInstanceName

This optional attribute specifies a name for the applet instance, which makes it possible for applets on the same page to find (and communicate with) each other.

width=pixels

height = pixels

These required attributes give the initial width and height (in pixels) of the applet display area, not counting any windows or dialogs that the applet brings up.

align = alignment

This attribute specifies the alignment of the applet. This attribute is defined in exactly the same way as the `IMG` element. The permitted values are: `top`, `middle`, `bottom`, `left` and `right`. The default is `bottom`.

vspace=pixels
hspace = pixels

These optional attributes specify the number of pixels above and below the applet (**VSPACE**) and on each side of the applet (**HSPACE**). They're treated the same way as the **IMG** element's **VSPACE** and **HSPACE** attributes.

The **PARAM** element is used to pass named parameters to applet:

<**PARAM NAME** = appletParameter **VALUE** = value>

PARAM elements are the only way to specify applet-specific parameters. Applets read user-specified values for parameters with the *getParameter()* method.

name = applet parameter name

value = parameter value

SGML character entities such as *´*; and *¹*; are expanded before the parameter value is passed to the applet. To include an **&** character use *&*;

Note: **PARAM** elements should be placed at the start of the content for the **APPLET** element.

Font

Requires start and end tags. This allows you to change the font size and/or color for the enclosed text. The attributes are: **SIZE** and **COLOR**. Font sizes are given in terms of a scalar range defined by the user agent with no direct mapping to point sizes etc. The **FONT** element may be phased out in future revisions to HTML.

size

This sets the font size for the contents of the font element. You can set **size** to an integer ranging from 1 to 7 for an absolute font size, or specify a relative font size with a signed integer value, e.g. **size**="+1" or **size**="-2". This is mapped to an absolute font size by adding the current base font size as set by the **BASEFONT** element (see below).

color

Used to set the color to stroke the text. Colors are given as RGB in hexadecimal notation or as one of 16 widely understood color names defined as per the BGCOLOR attribute on the BODY element.

Some user agents also support a *FACE* attribute which accepts a comma separated list of font names in order of preference. This is used to search for an installed font with the corresponding name. *FACE* is not part of HTML 3.2.

The following shows the effects of setting font to absolute sizes:

size=1 size=2 size=3 size=4 size=5 size=6 size=7

The following shows the effect of relative font sizes using a base font size of 3:

size=-4 size=-3 size=-2 size=-1 size=+1 size=+2 size=+3 size=+4

The same thing with a base font size of 6:

size=-4 size=-3 size=-2 size=-1 size=+1 size=+2
size=+3 size=+4

Basefont

Used to set the base font size. BASEFONT is an empty element so the end tag is forbidden. The SIZE attribute is an integer value ranging from 1 to 7. The base font size applies to the normal and preformatted text but not to headings, except where these are modified using the FONT element with a relative font size.

Br

Used to force a line break. This is an empty element so the end tag is forbidden. The CLEAR attribute can be used to move down past floating images on either margin. <BR CLEAR=LEFT> moves down past floating images on the left margin, <BR CLEAR=RIGHT> does the same for floating images on the right margin, while <BR CLEAR=ALL> does the same for such images on both left and right margins.

Map

The `MAP` element provides a mechanism for client-side image maps. These can be placed in the same document or grouped in a separate document although this isn't yet widely supported. The `MAP` element requires start and end tags. It contains one or more `AREA` elements that specify hotzones on the associated image and bind these hotzones to URLs.

Here is a simple example for a graphical navigational toolbar:

```


<map name="map1">
  <area href=guide.html alt="Access Guide" shape=rect coords="0,0,118,28">
  <area href=search.html alt="Search" shape=rect coords="184,0,276,28">
  <area href=shortcut.html alt="Go" shape=rect coords="118,0,184,28">
  <area href=top10.html alt="Top Ten" shape=rect coords="276,0,373,28">
</map>
```

The `MAP` element has one attribute `NAME` which is used to associate a name with a map. This is then used by the `USEMAP` attribute on the `IMG` element to reference the map via a URL fragment identifier. Note that the value of the `NAME` attribute is case sensitive.

The `AREA` element is an empty element and so the end tag is forbidden. It takes the following attributes: `SHAPE`, `COORDS`, `HREF`, `NOHREF` and `ALT`. The `SHAPE` and `COORDS` attributes define a region on the image. If the `SHAPE` attribute is omitted, `SHAPE="RECT"` is assumed.

```
shape=rect coords="left-x, top-y, right-x, bottom-y"
shape=circle coords="center-x, center-y, radius"
shape=poly coords="x1,y1, x2,y2, x3,y3, ..."
```

Where `x` and `y` are measured in pixels from the left/top of the associated image. If `x` and `y` values are given with a percent sign as a suffix, the values should be interpreted as percentages of the image's width and height, respectively. For example:

```
SHAPE=RECT COORDS="0, 0, 50%, 100%"
```

The `HREF` attribute gives a URL for the target of the hypertext link. The `NOHREF` attribute is used when you want to define a region that doesn't act as a hotzone. This is useful when you want to cut a hole in an underlying region acting as a hotzone.

If two or more regions overlap, the region defined first in the map definition takes precedence over subsequent regions. This means that `AREA` elements with `NOHREF` should generally be placed before ones with the `HREF` attribute.

The ALT attribute is used to provide text labels which can be displayed in the status line as the mouse or other pointing device is moved over hotzones, or for constructing a textual menu for non-graphical user agents. Authors are strongly recommended to provide meaningful ALT attributes to support interoperability with speech-based or text-only user agents.

Character Entities for ISO Latin-1

The following table provides a reference for those who want to include special characters in their html text, like á, ü or ê. The way to do so is to include in the html code either the keyword *&name*, where *name* stands for the name of any of the special characters as specified in the table below, or the keyword *&#* plus the corresponding number which you can find in the table. For instance, the code for á is either *á* or *á*.

```
<!-- (C) International Organization for Standardization 1986
      Permission to copy in any form is granted for use with
      conforming SGML systems and applications as defined in
      ISO 8879, provided this notice is included in all copies.
      This has been extended for use with HTML to cover the full
      set of codes in the range 160-255 decimal.
-->
<!-- Character entity set. Typical invocation:
      <!ENTITY % ISolat1 PUBLIC
           "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML">
      %ISolat1;
-->
<!ENTITY nbsp      CDATA "&#160;" -- no-break space -->
<!ENTITY iexcl    CDATA "&#161;" -- inverted exclamation mark -->
<!ENTITY cent     CDATA "&#162;" -- cent sign -->
<!ENTITY pound    CDATA "&#163;" -- pound sterling sign -->
<!ENTITY curren   CDATA "&#164;" -- general currency sign -->
<!ENTITY yen      CDATA "&#165;" -- yen sign -->
<!ENTITY brvbar   CDATA "&#166;" -- broken (vertical) bar -->
<!ENTITY sect     CDATA "&#167;" -- section sign -->
<!ENTITY uml      CDATA "&#168;" -- umlaut (dieresis) -->
<!ENTITY copy     CDATA "&#169;" -- copyright sign -->
<!ENTITY ordf     CDATA "&#170;" -- ordinal indicator, feminine -->
<!ENTITY laquo    CDATA "&#171;" -- angle quotation mark, left -->
<!ENTITY not      CDATA "&#172;" -- not sign -->
<!ENTITY shy      CDATA "&#173;" -- soft hyphen -->
<!ENTITY reg      CDATA "&#174;" -- registered sign -->
<!ENTITY macr     CDATA "&#175;" -- macron -->
<!ENTITY deg      CDATA "&#176;" -- degree sign -->
<!ENTITY plusmn   CDATA "&#177;" -- plus-or-minus sign -->
<!ENTITY sup2     CDATA "&#178;" -- superscript two -->
<!ENTITY sup3     CDATA "&#179;" -- superscript three -->
<!ENTITY acute    CDATA "&#180;" -- acute accent -->
<!ENTITY micro    CDATA "&#181;" -- micro sign -->
<!ENTITY para     CDATA "&#182;" -- pilcrow (paragraph sign) -->
<!ENTITY middot   CDATA "&#183;" -- middle dot -->
<!ENTITY cedil    CDATA "&#184;" -- cedilla -->
<!ENTITY sup1     CDATA "&#185;" -- superscript one -->
```

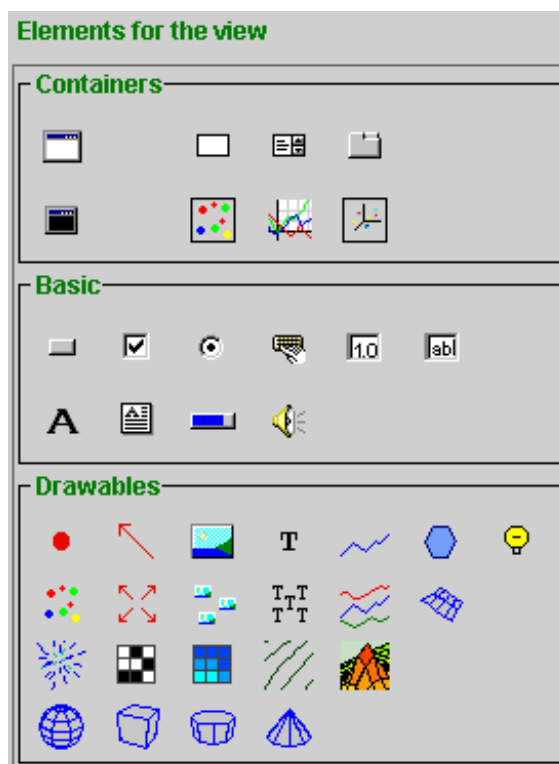
```
<!ENTITY ordm CDATA "&#186;" -- ordinal indicator, masculine -->
<!ENTITY raquo CDATA "&#187;" -- angle quotation mark, right -->
<!ENTITY frac14 CDATA "&#188;" -- fraction one-quarter -->
<!ENTITY frac12 CDATA "&#189;" -- fraction one-half -->
<!ENTITY frac34 CDATA "&#190;" -- fraction three-quarters -->
<!ENTITY iquest CDATA "&#191;" -- inverted question mark -->
<!ENTITY Agrave CDATA "&#192;" -- capital A, grave accent -->
<!ENTITY Aacute CDATA "&#193;" -- capital A, acute accent -->
<!ENTITY Acirc CDATA "&#194;" -- capital A, circumflex accent -->
<!ENTITY Atilde CDATA "&#195;" -- capital A, tilde -->
<!ENTITY Auml CDATA "&#196;" -- capital A, dieresis or umlaut
mark -->
<!ENTITY Aring CDATA "&#197;" -- capital A, ring -->
<!ENTITY AElig CDATA "&#198;" -- capital AE diphthong (ligature)
-->
<!ENTITY Ccedil CDATA "&#199;" -- capital C, cedilla -->
<!ENTITY Egrave CDATA "&#200;" -- capital E, grave accent -->
<!ENTITY Eacute CDATA "&#201;" -- capital E, acute accent -->
<!ENTITY Ecirc CDATA "&#202;" -- capital E, circumflex accent -->
<!ENTITY Euml CDATA "&#203;" -- capital E, dieresis or umlaut
mark -->
<!ENTITY Igrave CDATA "&#204;" -- capital I, grave accent -->
<!ENTITY Iacute CDATA "&#205;" -- capital I, acute accent -->
<!ENTITY Icirc CDATA "&#206;" -- capital I, circumflex accent -->
<!ENTITY Iuml CDATA "&#207;" -- capital I, dieresis or umlaut
mark -->
<!ENTITY ETH CDATA "&#208;" -- capital Eth, Icelandic -->
<!ENTITY Ntilde CDATA "&#209;" -- capital N, tilde -->
<!ENTITY Ograve CDATA "&#210;" -- capital O, grave accent -->
<!ENTITY Oacute CDATA "&#211;" -- capital O, acute accent -->
<!ENTITY Ocirc CDATA "&#212;" -- capital O, circumflex accent -->
<!ENTITY Otilde CDATA "&#213;" -- capital O, tilde -->
<!ENTITY Ouml CDATA "&#214;" -- capital O, dieresis or umlaut
mark -->
<!ENTITY times CDATA "&#215;" -- multiply sign -->
<!ENTITY Oslash CDATA "&#216;" -- capital O, slash -->
<!ENTITY Ugrave CDATA "&#217;" -- capital U, grave accent -->
<!ENTITY Uacute CDATA "&#218;" -- capital U, acute accent -->
<!ENTITY Ucirc CDATA "&#219;" -- capital U, circumflex accent -->
<!ENTITY Uuml CDATA "&#220;" -- capital U, dieresis or umlaut
mark -->
<!ENTITY Yacute CDATA "&#221;" -- capital Y, acute accent -->
<!ENTITY THORN CDATA "&#222;" -- capital THORN, Icelandic -->
<!ENTITY szlig CDATA "&#223;" -- small sharp s, German (sz
ligature) -->
<!ENTITY agrave CDATA "&#224;" -- small a, grave accent -->
<!ENTITY aacute CDATA "&#225;" -- small a, acute accent -->
<!ENTITY acirc CDATA "&#226;" -- small a, circumflex accent -->
<!ENTITY atilde CDATA "&#227;" -- small a, tilde -->
<!ENTITY auml CDATA "&#228;" -- small a, dieresis or umlaut mark
-->
<!ENTITY aring CDATA "&#229;" -- small a, ring -->
<!ENTITY aelig CDATA "&#230;" -- small ae diphthong (ligature) --
>
<!ENTITY ccedil CDATA "&#231;" -- small c, cedilla -->
<!ENTITY egrave CDATA "&#232;" -- small e, grave accent -->
<!ENTITY eacute CDATA "&#233;" -- small e, acute accent -->
<!ENTITY ecirc CDATA "&#234;" -- small e, circumflex accent -->
<!ENTITY euml CDATA "&#235;" -- small e, dieresis or umlaut mark
-->
<!ENTITY igrave CDATA "&#236;" -- small i, grave accent -->
```

```
<!ENTITY iacute CDATA "&#237;" -- small i, acute accent -->
<!ENTITY icirc  CDATA "&#238;" -- small i, circumflex accent -->
<!ENTITY iuml   CDATA "&#239;" -- small i, dieresis or umlaut mark
-->
<!ENTITY eth    CDATA "&#240;" -- small eth, Icelandic -->
<!ENTITY ntilde CDATA "&#241;" -- small n, tilde -->
<!ENTITY ograve CDATA "&#242;" -- small o, grave accent -->
<!ENTITY oacute CDATA "&#243;" -- small o, acute accent -->
<!ENTITY ocirc  CDATA "&#244;" -- small o, circumflex accent -->
<!ENTITY otilde CDATA "&#245;" -- small o, tilde -->
<!ENTITY ouml   CDATA "&#246;" -- small o, dieresis or umlaut mark
-->
<!ENTITY divide CDATA "&#247;" -- divide sign -->
<!ENTITY oslash CDATA "&#248;" -- small o, slash -->
<!ENTITY ugrave CDATA "&#249;" -- small u, grave accent -->
<!ENTITY uacute CDATA "&#250;" -- small u, acute accent -->
<!ENTITY ucirc  CDATA "&#251;" -- small u, circumflex accent -->
<!ENTITY uuml   CDATA "&#252;" -- small u, dieresis or umlaut mark
-->
<!ENTITY yacute CDATA "&#253;" -- small y, acute accent -->
<!ENTITY thorn  CDATA "&#254;" -- small thorn, Icelandic -->
<!ENTITY yuml   CDATA "&#255;" -- small y, dieresis or umlaut mark
-->
```

D. Reference pages for elements for the view

This appendix provides a reference page for each class of elements that can be used to build a view within **Ejs**. Each reference page contains a description of the element class and its main uses, as well as a table that lists all the properties for elements in this class.

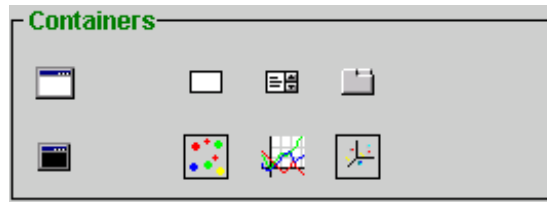
The elements are grouped by category and presented in the same order as they appear in **Ejs** view panel.





However, the actual number of elements that you may find in your copy of **Ejs** can vary from what is listed here. If you find that you have some that are not included here, this surely means that you have a later release of the software (that includes new elements) and you should update this appendix from the sever <http://fem.um.es/Ejs>.

If you see elements listed here that do not appear in your copy of **Ejs**, this can also mean that the software has been configured to provide a reduced set of elements in order to simplify its use. Ask your administrator about it, if you need any of these other elements.

Containers

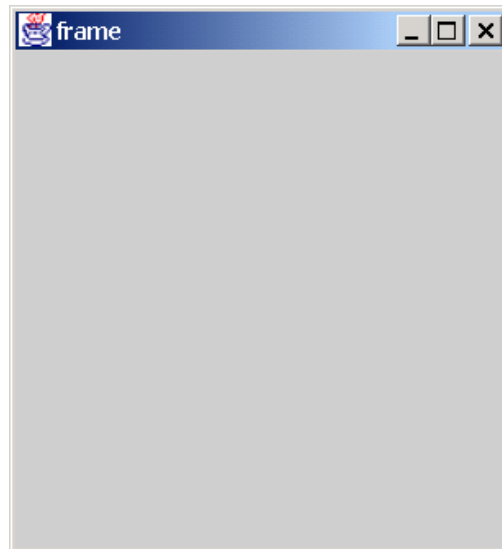


Frame

Icon :  ( when it is the main window)

Caption: A top level window

Description: A *Frame* is a container element that displays in a separate window. When a frame is closed (closed, not minimized) the simulation exits.



Frames have an internal boolean variable that corresponds to its visibility on the screen (that is, it is *true* if the frame is shown and *false* if it is hidden). Since the user can hide the dialog by clicking on its window icon controls, this allows the user to change the value of the variable.

The typical use of this variable is to associate it with the variable of a *Checkbox* basic element so that a whole window can be shown/hidden when the checkbox is selected/unselected.

Frames trigger no action but, as said above, they have the property to exit the application when they are closed.

Table of Properties		
Name	Description	Possible values
Title	The text to be displayed as the title for the frame's window	Constant: Any string is valid, spaces are allowed between words, but are trimmed from the beginning and the end of the text Variable: A variable of type <i>String</i>
Layout	The layout policy for the frame, see chapter 6 of the manual	Constant: One of the following: <i>border</i> , <i>flow</i> , <i>grid</i> , <i>hbox</i> or <i>vbox</i> . <ul style="list-style-type: none"> <i>border</i> accepts two optional parameters, the horizontal and

		<p>vertical separation between childrens. Hence, both <i>border</i> and <i>border:h,v</i> (<i>h</i> and <i>v</i> in pixels) are valid</p> <ul style="list-style-type: none"> • <i>flow</i> requires an extra parameter, <i>flow:align</i>, where <i>align</i> is the desired horizontal alignment for the children: either <i>left</i>, <i>center</i> or <i>right</i>. It also accepts two optional parameters for the gaps between children. • <i>Grid</i> requires two parameters, <i>grid:x,y</i> (where <i>x</i> is the number of rows and <i>y</i> the number of columns). It also accepts two extra parameters for the gaps between children. <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Layout</i></p>
Location	The position of the frame in the screen	<p>Constant: The <i>x</i> and <i>y</i> integer screen coordinates of the upper-left corner, separated by commas. For example, <i>0,0</i> sets the frame at the upper-left corner of the screen.</p> <p>The special value <i>center</i>, places the frame at the center of the screen.</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Point</i></p>
Visible	The visibility of the frame	<p>Constant: Either <i>true</i> or <i>false</i>.</p> <p>Variable: A variable of type <i>boolean</i></p>
Size	The size of the frame in the screen	<p>Constant: The width and height integer dimensions in screen coordinates, separated by a comma. For example, <i>200,200</i> sets a squared frame of 200x200 pixels</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i></p>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	<p>Constant: One of the following basic color names: <i>black</i>, <i>blue</i>, <i>cyan</i>, <i>darkGray</i>, <i>gray</i>, <i>green</i>, <i>lightGray</i>, <i>magenta</i>, <i>orange</i>, <i>pink</i>, <i>red</i>, <i>white</i>, <i>yellow</i>.</p> <p>Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i></p>

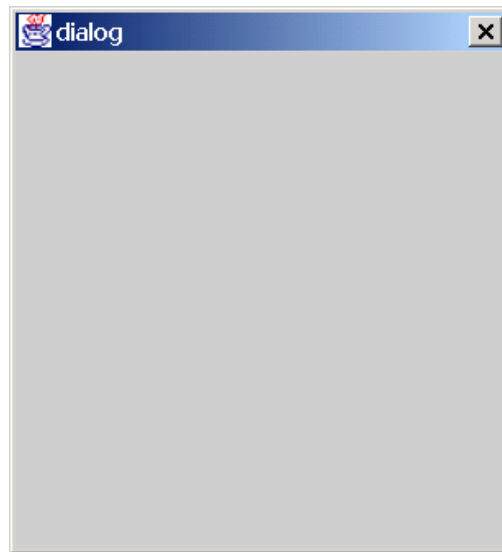
Background	The color used for the background of the element and for its children (unless children set their own value)	See <i>Foreground</i> above
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain</i> , <i>bold</i> , <i>italic</i> , <i>bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>

Dialog

Icon : 

Caption: A dialog window

Description: A *Dialog* is a container element that displays in a separate window. Unlike frames, closing a dialog does not exit the simulation. Dialogs have the peculiarity that they always show on top of the last frame that has been created before them (if there is any). That is, the previous frame can not hide the dialog.



Dialogs have an internal boolean variable that corresponds to its visibility on the screen (that is, it is *true* if the dialog is shown and *false* if it is hidden). Since the user can hide the dialog by clicking on its window icon controls, this allows the user to change the value of the variable.

The typical use of this variable is to associate it with the variable of a *Checkbox* basic element so that a whole window can be shown/hidden when the checkbox is selected/unselected.

Dialogs trigger no action.

Table of Properties		
Name	Description	Possible values
Title	The text to be displayed as the title for the dialog's window	Constant: Any string is valid, spaces are allowed between words, but are trimmed from the beginning and the end of the text Variable: A variable of type <i>String</i>
Layout	The layout policy for the dialog, see chapter 6 of the manual	Constant: One of the following: <i>border</i> , <i>flow</i> , <i>grid</i> , <i>hbox</i> or <i>vbox</i> .

		<ul style="list-style-type: none"> • <i>border</i> accepts two optional parameters, the horizontal and vertical separation between childrens. Hence, both <i>border</i> and <i>border:h,v</i> (<i>h</i> and <i>v</i> in pixels) are valid • <i>flow</i> requires an extra parameter, <i>flow:align</i>, where <i>align</i> is the desired horizontal alignment for the children: either <i>left</i>, <i>center</i> or <i>right</i>. It also accepts two optional parameters for the gaps between children. • <i>Grid</i> requires two parameters, <i>grid:x,y</i> (where <i>x</i> is the number of rows and <i>y</i> the number of columns). It also accepts two extra parameters for the gaps between children. <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Layout</i></p>
Location	The position of the dialog in the screen	<p>Constant: The x and y integer screen coordinates of the upper-left corner, separated by a comma. The special value <i>center</i>, places the dialog at the center of the screen.</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Point</i></p>
Visible	The visibility of the dialog	<p>Constant: Either <i>true</i> or <i>false</i>.</p> <p>Variable: A variable of type <i>boolean</i></p>
Size	The size of the dialog in the screen	<p>Constant: The width and height integer dimensions in screen coordinates, separated by commas. For example, <i>200,200</i> sets a squared dialog of 200x200 pixels</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i></p>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	<p>Constant: One of the following basic color names: <i>black</i>, <i>blue</i>, <i>cyan</i>, <i>darkGray</i>, <i>gray</i>, <i>green</i>, <i>lightGray</i>, <i>magenta</i>, <i>orange</i>, <i>pink</i>, <i>red</i>, <i>white</i>, <i>yellow</i>.</p> <p>Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i></p>

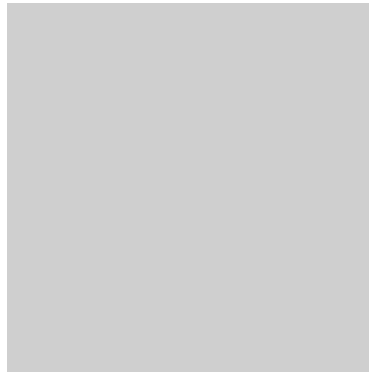
Background	The color used for the background of the element and for its children (unless children set their own value)	See <i>Foreground</i> above
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain</i> , <i>bold</i> , <i>italic</i> , <i>bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>

Panel

Icon :

Caption: A basic container panel

Description: A *Panel* is a basic container element. It is used to accommodate children according to a given layout policy, as specified by its *Layout* property.



A panel can be made visible or hidden by modifying its *Visible* property. This is useful to show/hide certain elements from the view according to the simulation's logic. The panel cannot change the value of this variable directly.

Panels trigger no action.

Table of Properties		
Name	Description	Possible values
Layout	The layout policy for the panel, see chapter 6 of the manual	<p>Constant: One of the following: <i>border</i>, <i>flow</i>, <i>grid</i>, <i>hbox</i> or <i>vbox</i>.</p> <ul style="list-style-type: none"> • <i>border</i> accepts two optional parameters, the horizontal and vertical separation between childrens. Hence, both <i>border</i> and <i>border:h,v</i> (<i>h</i> and <i>v</i> in pixels) are valid • <i>flow</i> requires an extra parameter, <i>flow:align</i>, where <i>align</i> is the desired horizontal alignment for the children: either <i>left</i>, <i>center</i> or <i>right</i>. It also accepts two optional parameters for the gaps between children. • <i>Grid</i> requires two parameters, <i>grid:x,y</i> (where <i>x</i> is the number of rows and <i>y</i> the number of columns). It also accepts two extra parameters for the gaps between

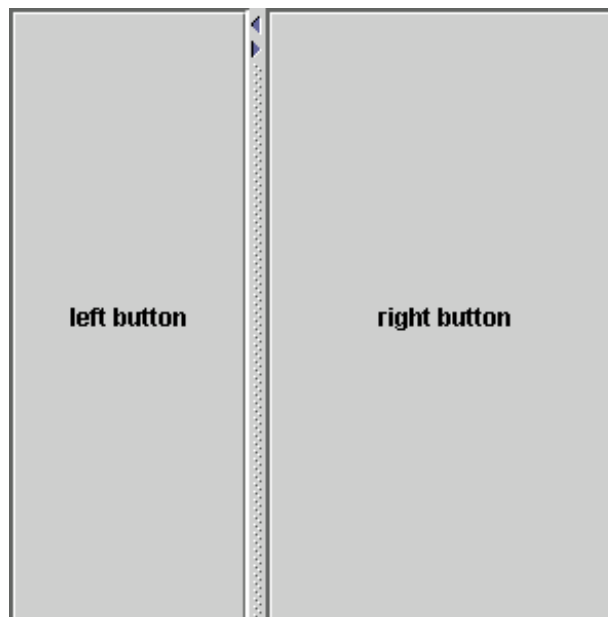
		<p>children.</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Layout</i></p>
Border	An empty area surrounding the panel	<p>Constant: The top, left, bottom and right space, in pixels, separated by commas. For example, <i>5,10,5,10</i> leaves 5 pixels in the vertical margins and 10 in the horizontal ones</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Rectangle</i></p>
Visible	The visibility of the panel	<p>Constant: Either <i>true</i> or <i>false</i>.</p> <p>Variable: A variable of type <i>boolean</i></p>
Size	The preferred size for the panel. Parents can modify this, according to their layout	<p>Constant: The width and height integer dimensions in screen coordinates, separated by commas</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i></p>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	<p>Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i>.</p> <p>Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i></p>
Background	The color used for the background of the element and for its children (unless children set their own value)	See <i>Foreground</i> above
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	<p>Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i>. Example: <i>Monospaced,italic,18</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i></p>
Tooltip	The text displayed when the cursor lingers over the element	<p>Constant: Any string (of reasonable length)</p> <p>Variable: A variable of type <i>String</i></p>

SplitPanel

Icon : 

Caption: A container with two separated areas

Description: A *SplitPanel* is a container element that can hold up to two children. It separates its screen area in two parts, either horizontally or vertically, and assigns each of these parts to its children. It can also display a divider that can be used to dynamically resize the children.



Similar to basic panels, a split panel can be made visible or hidden by modifying its *Visible* property. This is useful to show/hide certain elements from the view according to the simulation's logic. The split panel cannot change the value of this variable directly.

Split panels trigger no action.

Table of Properties		
Name	Description	Possible values
Orientation	The direction in which to establish the separation	Constant: Either <i>horizontal</i> or <i>vertical</i> Variable: A variable of type <i>int</i>
One touch	Whether it should provide a UI widget to collapse/expand the divider	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Visible	The visibility of the panel	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Size	The preferred size for the panel. Parents can modify this,	Constant: The width and height integer dimensions in screen

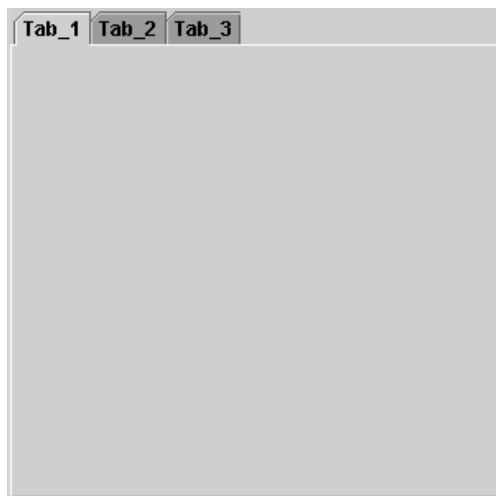
	according to their layout	coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element and for its children (unless children set their own value)	See <i>Foreground</i> above
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

TabbedPanel

Icon : 

Caption: A container with tabs

Description: A *TabbedPanel* is a container element that accommodates children one on top of the other and provides a tab system that allows the user to select (by clicking on the corresponding tab) which of the children must be visible at a given moment. Tabs display the name of the child.



Similar to basic panels, a tabbed panel can be made visible or hidden by modifying its *Visible* property. This is useful to show/hide certain elements from the view according to the simulation's logic. The tabbed panel cannot change the value of this variable directly.

Tabbed panels trigger no action.

Table of Properties		
Name	Description	Possible values
Tab Pos	Where to place the tabs	Constant: Either <i>top</i> , <i>bottom</i> , <i>left</i> or <i>right</i> Variable: A variable of type <i>int</i>
Visible	The visibility of the panel	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Size	The preferred size for the panel. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> ,

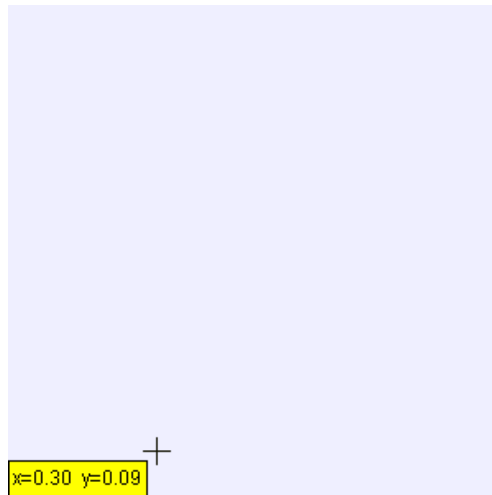
		<p><i>yellow</i>. Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i></p>
Background	The color used for the background of the element and for its children (unless children set their own value)	See <i>Foreground</i> above
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	<p>Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i>. Example: <i>Monospaced,italic,18</i>. The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i></p>
Tooltip	The text displayed when the cursor lingers over the element	<p>Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i></p>

DrawingPanel

Icon : 

Caption: A 2D container for drawables

Description: A *DrawingPanel* is a special container element that accommodates children of the *Drawables* set (see chapter 6).



Drawing panels draw in the rectangular region of the (2D) plane which goes from the point given by the coordinates (*Minimum X*, *Minimum Y*)¹ to the point (*Maximum X*, *Maximum Y*)¹, although they can also be instructed to automatically compute the scales on each, or both, of the X and Y axes so that they will show all their children elements.

Drawing panels are interactive and respond to different gestures of the mouse over it. The sequence is as follows:

When the user clicks on the panel, the action indicated by the *On Press* property is called. Immediately after, the properties *X* and *Y* are set to the mouse position (in the panel's own real coordinates), which also triggers the action linked to the *On Drag* property.

When the user is dragged (with the mouse button hold down) the properties *X* and *Y* are updated to the mouse position and the action linked to *On Drag* is triggered.

When the user releases the mouse button (if he or she does it inside the element) the action linked to the property *On Release* is triggered.

¹ Minimum X, Minimum Y, etc. are properties of this element.

Although a drawing panel belongs to the group of containers, it should not be used to host children other than from the group of drawables.

Table of Properties		
Name	Description	Possible values
Autoscale X	Whether to automatically compute the scale for the X coordinates	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale Y	Whether to automatically compute the scale for the Y coordinates	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Minimum X	The minimum X coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum X	The maximum X coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Minimum Y	The minimum Y coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum Y	The maximum Y coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
X	The X coordinate of the mouse	Constant: not applicable Variable: A variable of type <i>int</i> or <i>double</i>
Y	The Y coordinate of the mouse	Constant: not applicable Variable: A variable of type <i>int</i> or <i>double</i>
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action
Coordinates	Whether it should display the coordinates when the mouse is clicked on the element	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Square	Whether to keep a squared aspect ratio. This can modify the extrema for the axes	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Gutters	The gutters (unused space) around the drawing area	Constant: The top, left, bottom and right space, in pixels, separated by commas Variable: An <i>Object</i> variable of the class <i>java.awt.Rectangle</i>
Size	The preferred size for the panel.	Constant: The width and height

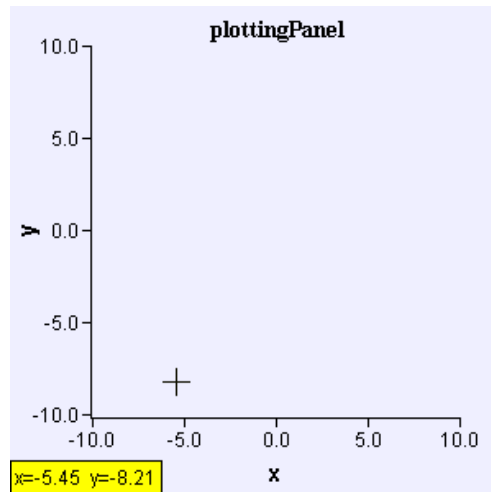
	Parents can modify this, according to their layout	integer dimensions in screen coordinates, separated by a commas Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element and for its children (unless children set their own value)	See <i>Foreground</i> above
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

PlottingPanel

Icon : 

Caption: A 2D container to plot series of (x,y) points

Description: A *PlottingPanel* is a special *DrawingPanel* that includes, by default, the display of two axes, in the X and Y direction, as well as titles for both axes and for the panel itself.



Since plotting panels' main purpose is the display of drawables of the *Dataset* class, which are not responsive to user interaction, plotting panels have suppressed the interaction properties of drawing panels (*X*, *Y*, and the actions *On Press*, *On Drag*, *On Release*).

Table of Properties		
Name	Description	Possible values
Title	A text to be displayed at the top of the panel	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
X Axis	The position in which to place the X axis	Constant: Any constant number. By default, the axis is drawn at the bottom margin (see Gutters) of the plot. Variable: A variable of type <i>int</i> or <i>double</i>
Title X	A text to be displayed on the X axis	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
Y axis	The position in which to place the Y axis	Constant: Any constant number. By default, the axis is drawn at the left margin (See Gutters) of the plot. Variable: A variable of type <i>int</i> or <i>double</i>
Title Y	A text to be displayed on the Y	Constant: Any string (of reasonable

	axis	length) Variable: A variable of type <i>String</i>
Autoscale X	Whether to automatically compute the scale for the X coordinates	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale Y	Whether to automatically compute the scale for the Y coordinates	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Minimum X	The minimum X coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum X	The maximum X coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Minimum Y	The minimum Y coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum Y	The maximum Y coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Coordinates	Whether it should display the coordinates when the mouse is clicked on the element	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Square	Whether to keep a squared aspect ratio. This can modify the extrema for the axes	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Gutters	The gutters (unused space) around the drawing area	Constant: The top, left, bottom and right space, in pixels, separated by commas Variable: An <i>Object</i> variable of the class <i>java.awt.Rectangle</i>
Size	The preferred size for the panel. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the	See <i>Foreground</i> above

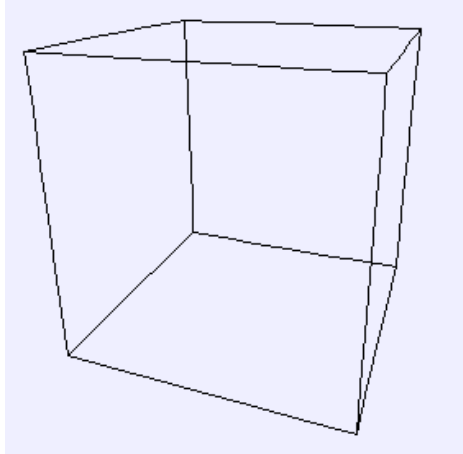
	background of the element and for its children (unless children set their own value)	
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain</i> , <i>bold</i> , <i>italic</i> , <i>bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

DrawingPanel3D

Icon : 

Caption: A 3D container for drawables

Description: A *DrawingPanel3D* is a special, 3D enabled, container element that accommodates children of the *Drawables* set (see chapter 6).



Drawing 3D panels draw in the three dimensional region of the space which goes from the point given by the coordinates (*Minimum X, Minimum Y, Minimum Z*) to the point (*Maximum X, Maximum Y, Maximum Z*)², although they can also be instructed to automatically compute the scales on each, or all, of the axes so that they will show all their children elements.

Drawing 3D panels are endowed with a ‘not too sophisticated’ capability of removing hidden lines, which can improve the visibility of the objects inside them.

Drawing 3D panels are not interactive in the same way Drawing 2D panels are. However they respond to mouse interaction:

- Clicking and dragging on the panel changes the perspective point of view.
- If the ‘Control’ key is pressed while the mouse is operated, then the scene is panned.
- If the ‘Shift’ key is pressed, then the scene is zoomed in or out, depending on the mouse motion.

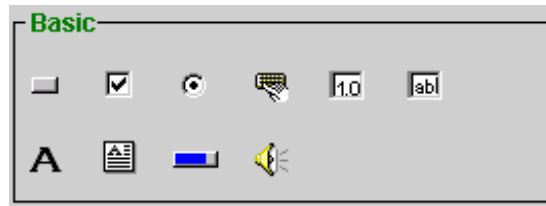
Although a drawing 3D panel belongs to the group of containers, it should not be used to host children other than from the group of drawables.

² Minimum X, Minimum Y, etc. are properties of this element.

Table of Properties		
Name	Description	Possible values
Autoscale X	Whether to automatically compute the scale for the X coordinates	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale Y	Whether to automatically compute the scale for the Y coordinates	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale Z	Whether to automatically compute the scale for the Z coordinates	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Minimum X	The minimum X coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum X	The maximum X coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Minimum Y	The minimum Y coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum Y	The maximum Y coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Minimum Z	The minimum Z coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum Z	The maximum Z coordinate that can be drawn in the panel	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Alpha	The horizontal angle (in degrees) to rotate the view before projecting to the screen	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Beta	The vertical angle (in degrees) to rotate the view before projecting to the screen	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Zoom	The magnifying factor. A factor of 1.0 leaves the scene unmodified	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Perspective	Whether it should apply a conic perspective (objects farther away look smaller and dimmer)	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Show Box	Whether it should display a bounding box	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Show Axes	Whether it should display the axes	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Hide Lines	Whether it should remove hidden lines	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Size	The preferred size for the panel. Parents can modify this,	Constant: The width and height integer dimensions in screen

	according to their layout	coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used for the foreground of the element and for its children (unless children set their own value)	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element and for its children (unless children set their own value)	See <i>Foreground</i> above
Font	The font to be used by any text displayed by the element and by its children (unless children set their own value)	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

Basic elements



Button

Icon : 

Caption: A button for actions

Description: A *Button* is a basic element that is used to trigger an action. It displays a text or an image, or both, and triggers the associated action when the button is clicked (that is, pressed and released).



Buttons can be disabled (that is, the user can click on them but they will not respond) by setting its *Enabled* property to false. In this case, the button interface is grayed out.

Table of Properties		
Name	Description	Possible values
Text	The text displayed by the button	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
Image	A gif file that holds the image for the button. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Alignment	The horizontal alignment of the icon and text	Constant: either <i>left</i> , <i>center</i> , <i>right</i> (the default), <i>leading</i> or <i>trailing</i> . Variable: A variable of type <i>int</i>
Action	The action to trigger when the button is clicked	Constant: not applicable Variable: An action
Enabled	Whether the button can be clicked or not	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from

		0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

Checkbox

Icon : 

Caption: A check box for boolean values

Description: A *Checkbox* is a basic element that is used to display and modify a boolean value.



Check boxes can trigger an action whenever they are clicked upon them, either to select or to unselect them. Besides this, a second action can be triggered in any of the two cases, separately; that is only when the element is selected or unselected. This second action (the one associated to the property *Action On* or *Action Off*, respectively) is triggered always after the first one (the one associated to the *Action* property).

Table of Properties		
Name	Description	Possible values
Text	The text displayed by the checkbox	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
Image	A gif file that holds the image for the checkbox. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Selected Image	A gif file that will be displayed when the element is in selected state. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Alignment	The horizontal alignment of the icon and text	Constant: either <i>left</i> , <i>center</i> , <i>right</i> (the default), <i>leading</i> or <i>trailing</i> . Variable: A variable of type <i>int</i>
Variable	The value to be displayed and modified	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Selected	The initial value for the variable	Constant: Either <i>true</i> or <i>false</i> . Variable: Not applicable
Action	The action to trigger when the element is clicked	Constant: not applicable Variable: An action
Action On	The action to trigger when the element is selected	Constant: not applicable Variable: An action
Action Off	The action to trigger when the element is unselected	Constant: not applicable Variable: An action
Enabled	Whether the button can be clicked or not	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>

Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

RadioButton

Icon : 

Caption: A radio button for boolean values

Description: A *RadioButton* is a basic element that is used to display and modify a boolean value. In this sense it works similarly to a *Checkbox*. The difference with these is that radio buttons work in groups. That is, when more than one radio buttons coexist in the same container, only one of them can be selected at a given moment (it is also possible that none of them is selected). Hence, if one is clicked upon, in order to select it, it automatically unselects the others.



Three radio buttons in a panel

Radio buttons trigger an action whenever they are clicked upon them, either to select or to unselect them. Besides this, a second action can be triggered in any of the two cases, separately; that is only when the element is selected or unselected. This second action (the one associated to the property *Action On* or *Action Off*, respectively) is triggered always after the first one (the one associated to the *Action* property).

If the button is unselected because any other radio button of its group is selected, the corresponding actions will not be triggered.

Table of Properties		
Name	Description	Possible values
Text	The text displayed by the button	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
Image	A gif file that holds the image for the button. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Selected Image	A gif file that will be displayed when the element is in selected state. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Alignment	The horizontal alignment of the icon and text	Constant: either <i>left</i> , <i>center</i> , <i>right</i> (the default), <i>leading</i> or <i>trailing</i> .

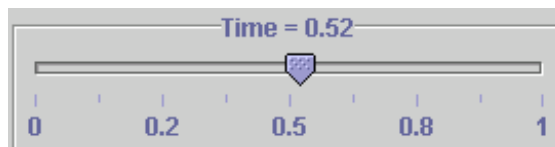
		Variable: A variable of type <i>int</i>
Variable	The value to be displayed and modified	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Selected	The initial value for the variable	Constant: Either <i>true</i> or <i>false</i> . Variable: Not applicable
Action	The action to trigger when the button is clicked	Constant: not applicable Variable: An action
Action On	The action to trigger when the element is selected	Constant: not applicable Variable: An action
Action Off	The action to trigger when the element is unselected	Constant: not applicable Variable: An action
Enabled	Whether the button can be clicked or not	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

Slider

Icon : 

Caption: A slider to display and modify a value

Description: A *Slider* is an element that displays the value of a numerical variable. The value is displayed positioning the knob of a slider between a *minimum* and a *maximum* value. Also, if the *format* property is set to a non-empty string, the value is displayed in a textual form at the top of the slider. The value can be edited by dragging a knob to a new position.



The element triggers the method indicated by the corresponding action property when the knob is pressed, dragged or released.

Tip: when using the a slider to display and edit an integer variable, it is convenient to set both *format* and *ticksFormat* to *0;-0* and *closest to true*.

Table of Properties		
Name	Description	Possible values
Variable	The value to be displayed and modified	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Value	The initial value for the variable	Constant: Any constant number Variable: Not applicable
Minimum	The minimum value displayed. If the variable is set to a smaller value, the slider displays this minimum	Constant: Any constant number. Default is 0.0 Variable: A variable of type <i>int</i> or <i>double</i>
Maximum	The maximum value displayed. If the variable is set to a bigger value, the slider displays this maximum	Constant: Any constant number. Default is 1.0 Variable: A variable of type <i>int</i> or <i>double</i>
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action
Format	The format used to display the	Constant: Any string valid for the

	value. If not set, the value won't be displayed	constructor of the class <i>java.text.DecimalFormat</i> (see appendix B) Variable: An <i>Object</i> variable of the class <i>java.text.DecimalFormat</i>
Enabled	Whether the knob can be dragged	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Ticks	The number of ticks to use between the <i>minimum</i> and the <i>maximum</i> . Only those at odd positions are labeled	Constant: Any (reasonable) integer number Variable: A variable of type <i>int</i>
Ticks Format	The format used to display the ticks. If not set, the ticks won't be displayed	Constant: Any string valid for the constructor of the class <i>java.text.DecimalFormat</i> (see appendix B) Variable: An <i>Object</i> variable of the class <i>java.text.DecimalFormat</i>
Closest	Whether the knob should resolve to the closest tick when released	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Orientation	Whether to display the slider horizontally or vertically	Constant: Either <i>horizontal</i> or <i>vertical</i> Variable: A variable of type <i>int</i>
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system

		Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

Field (or NumberField)

Icon : 

Caption: A text field to display and modify a number

Description: A *Field* or *NumberField* is an element that displays the value of a numerical variable. The value is displayed using a text field that can be edited to change the value of the variable.

Time = 0.52

When you start editing the field, it changes its background color to yellow. This helps you identify visually that it is displaying a new value but that this value has not yet been effectively entered. Only when you hit the return key is the value parsed in and the background changes back to the original one.

The element triggers the method indicated by the *Action* property when the return key is hit, thus assuming you have finished editing of the value.

Table of Properties		
Name	Description	Possible values
Variable	The value to be displayed and modified	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Value	The initial value for the variable	Constant: Any constant number Variable: Not applicable
Editable	Whether the value can be modified	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Format	The format used to display the value	Constant: Any string valid for the constructor of the class <i>java.text.DecimalFormat</i> (see appendix B). Default is <i>0.000;-0.000</i> Variable: An <i>Object</i> variable of the class <i>java.text.DecimalFormat</i>
Action	The action to trigger when the return key is hit	Constant: not applicable Variable: An action
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> .

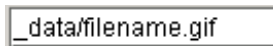
		<p>Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i></p>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	<p>Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i>. Example: <i>Monospaced,italic,18</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i></p>
Tooltip	The text displayed when the cursor lingers over the element	<p>Constant: Any string (of reasonable length)</p> <p>Variable: A variable of type <i>String</i></p>

TextField

Icon : 

Caption: A field to display and modify a string

Description: A *TextField* is an element that displays a String variable. The value is displayed using a text field that can be edited to change the value of the variable.



The element triggers the method indicated by the *Action* property when the return key is hit, thus assuming you have finished editing of the value.

When you start editing the field, it changes its background color to yellow. This helps you identify visually that it is displaying a new value but that this value has not yet been effectively entered. Only when you hit the return key is the value parsed in and the background changes back to the original one.

Table of Properties		
Name	Description	Possible values
Variable	The value to be displayed and modified	Constant: Any constant String Variable: A variable of type <i>String</i>
Value	The initial value for the variable	Constant: Any constant String Variable: Not applicable
Editable	Whether the value can be modified	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Action	The action to trigger when the return key is hit	Constant: not applicable Variable: An action
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system

		Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

Label

Icon : **A**

Caption: A decorative label

Description: A *Label* is a basic element that is used to display a decorative text or image, or both.



Labels can not be linked to variables nor can trigger any action.

Table of Properties		
Name	Description	Possible values
Text	The text displayed by the label	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
Image	A gif file that holds the image for the label. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Alignment	Sets the alignment of the label's contents along the X axis.	Constant: either <i>left</i> , <i>center</i> , <i>right</i> , <i>leading</i> or <i>trailing</i> . Variable: A variable of type <i>int</i>
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	Constant: The family name, style and

		size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

TextArea

Icon : 

Caption: A text area where to print

Description: A *TextArea* is an element that can be used to print textual messages according to the simulation's logic. If you want to print a message in any part of your simulation model, you need to include a text area element in your view and then you can use the sentence

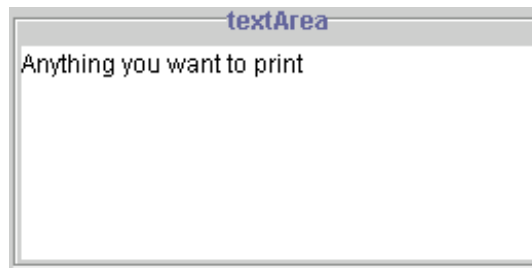
```
_print ("Anything you want to print");
```

or

```
_println ("Anything you want to print");
```

The message will appear in the text area. In the second case, a new line character is added to the message, hence subsequent messages will appear in a new line.

The predefined action *_clearView()* will, among other things, clear any text area in your view.



Only one such text area can be present in a given simulation view.

Text areas trigger no actions.

Table of Properties		
Name	Description	Possible values
Title	The title text that will appear at the top of the text area	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white,</i>

		<p><i>yellow</i>. Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i></p>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	<p>Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i>. Example: <i>Monospaced,italic,18</i>. The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i></p>
Tooltip	The text displayed when the cursor lingers over the element	<p>Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i></p>

Bar

Icon : 

Caption: A bar that displays a value

Description: A *Bar* is a basic element that is used to display a numeric value. The value is displayed using a progress bar between a *minimum* and a *maximum* value. Also, if the *format* property is set to a non-empty string, the value is displayed in a textual form in the center of the bar. The value can not be edited.



Bars trigger no action.

Table of Properties		
Name	Description	Possible values
Variable	The value to be displayed	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Minimum	The minimum value displayed. If the variable is set to a smaller value, the bar displays this minimum	Constant: Any constant number. Default is 0.0 Variable: A variable of type <i>int</i> or <i>double</i>
Maximum	The maximum value displayed. If the variable is set to a bigger value, the bar displays this maximum	Constant: Any constant number. Default is 1.0 Variable: A variable of type <i>int</i> or <i>double</i>
Format	The format used to display the value. If not set, the value won't be displayed	Constant: Any string valid for the constructor of the class <i>java.text.DecimalFormat</i> (see appendix B) Variable: An <i>Object</i> variable of the class <i>java.text.DecimalFormat</i>
Orientation	Whether to display the bar horizontally or vertically	Constant: Either <i>horizontal</i> or <i>vertical</i> Variable: A variable of type <i>int</i>
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> ,

		<p><i>magenta, orange, pink, red, white, yellow.</i></p> <p>Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i></p>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	<p>Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i>. Example: <i>Monospaced,italic,18</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i></p>
Tooltip	The text displayed when the cursor lingers over the element	<p>Constant: Any string (of reasonable length)</p> <p>Variable: A variable of type <i>String</i></p>

Sound

Icon : 

Caption: A sound-enabled checkbox

Description: *Sound* is a particular type of *Checkbox* which allows to play a sound according to the value of the internal boolean value. The sound is played continuously (in a loop) until the boolean value turns to be *false*.

For this reason, in order to let the user manually stop the sound (which could otherwise play to turn us all crazy!) this sound capability has been given to a check box.

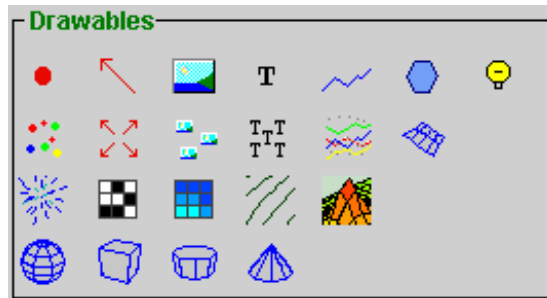


Sound elements can trigger an action whenever they are clicked upon them, either to select or to unselect them. Besides this, a second action can be triggered in any of the two cases, separately; that is only when the element is selected or unselected. This second action (the one associated to the property *Action On* or *Action Off*, respectively) is triggered always after the first one (the one associated to the *Action* property).

Table of Properties		
Name	Description	Possible values
Audio File	The audio file to play	Constant: The name of an existing AU, AIFF or WAV file. The file location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Text	The text displayed by the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
Image	A gif file that holds the image for the element. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Selected Image	A gif file that will be displayed when the element is in selected state. Animated gif are also possible	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
Alignment	The horizontal alignment of the icon and text	Constant: either <i>left</i> , <i>center</i> , <i>right</i> (the default), <i>leading</i> or <i>trailing</i> . Variable: A variable of type <i>int</i>

Variable	The value to be displayed and modified	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Selected	The initial value for the variable	Constant: Either <i>true</i> or <i>false</i> . Variable: Not applicable
Action	The action to trigger when the element is clicked	Constant: not applicable Variable: An action
Action On	The action to trigger when the element is selected	Constant: not applicable Variable: An action
Action Off	The action to trigger when the element is unselected	Constant: not applicable Variable: An action
Enabled	Whether the button can be clicked or not	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Size	The preferred size for the element. Parents can modify this, according to their layout	Constant: The width and height integer dimensions in screen coordinates, separated by a comma Variable: An <i>Object</i> variable of the class <i>java.awt.Dimension</i>
Foreground	The color used to display the text	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Background	The color used for the background of the element	See <i>Foreground</i> above
Font	The font used to display the text	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain, bold, italic, bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Tooltip	The text displayed when the cursor lingers over the element	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>

Drawables



Particle

Icon : 

Caption: An interactive particle

Description: A *Particle* is a drawable element that draws a simple geometric shape at a given location of its parent.



A filled-circle particle

The shape is drawn at the given location, with the specified size. However, a scale factor is also applied before drawing the element. This helps visualize elements which are of small size, relative to its parent's coordinate system. The location indicates the hot spot (or sensitive point) of the shape. However, the shape can be drawn in several different positions relative to this hot spot (see property *Position* in the table).

The element is interactive (if the parent is interactive) and triggers the method indicated by the corresponding action property when it is pressed, dragged or released.

Table of Properties		
Name	Description	Possible values
X	The X coordinate of the location of the particle	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Y	The Y coordinate of the location of the particle	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Z	The Z coordinate of the location of the particle	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size X	The X component of the size of the particle	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Y	The Y component of the size of the particle	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Z	The Z component of the size of the particle	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Enabled	Whether the element is	Constant: Either <i>true</i> or <i>false</i> .

	responsive to user interaction	Variable: A variable of type <i>boolean</i>
Scale X	A scale factor in the X axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Scale Y	A scale factor in the Y axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Scale Z	A scale factor in the Z axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Style	The type of shape to draw	Constant: Either a simple dot or a filled or hollow circle or square. The valid values are <i>NO_MARKER</i> , <i>CIRCLE</i> , <i>FILLED_CIRCLE</i> , <i>SQUARE</i> , <i>FILLED_SQUARE</i> Variable: A variable of type <i>int</i>
Position	The position of the shape relative to the hot spot	Constant: One of <i>CENTERED</i> , <i>HOR_CENTERED</i> , <i>VER_CENTERED</i> , <i>LOWER_LEFT</i> , <i>UPPER_LEFT</i> Variable: A variable of type <i>int</i>
Color	The color used to draw the element	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action

Arrow

Icon : 

Caption: An interactive vector (or line)

Description: An *Arrow* is a drawable element that draws a simple vector or line at a given location of its parent.



A vector arrow

The vector is drawn at the given location, with the specified size. However, a scale factor is also applied before drawing the element. This helps visualize elements which are of small size, relative to its parent's coordinate system. The location indicates the origin of the vector. Its hot spot (or sensitive point) is placed at the end point of the vector. Thus, dragging the mouse on the head, modifies the size of the vector.

The element is interactive (if the parent is interactive) and triggers the method indicated by the corresponding action property when it is pressed, dragged or released.

Table of Properties		
Name	Description	Possible values
X	The X coordinate of the origin of the vector	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Y	The Y coordinate of the origin of the vector	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Z	The Z coordinate of the origin of the vector	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size X	The X component of the size of the vector	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Y	The Y component of the size of the vector	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Z	The Z component of the size of the vector	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Enabled	Whether the element is	Constant: Either <i>true</i> or <i>false</i> .

	responsive to user interaction	Variable: A variable of type <i>boolean</i>
Scale X	A scale factor in the X axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Scale Y	A scale factor in the Y axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Scale Z	A scale factor in the Z axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Style	The type of vector to draw	Constant: Either an arrow, a segment of a segment with a square handle at the end point can be drawn. The valid values are <i>ARROW</i> , <i>SEGMENT</i> and <i>BOX</i> Variable: A variable of type <i>int</i>
Color	The color used to draw the element	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action

Image

Icon : 

Caption: An interactive gif image

Description: An *Image* is a drawable element that draws a gif image at a given location of its parent.



The image is drawn at the given location, with the specified size. However, a scale factor is also applied before drawing the element. This helps visualize elements which are of small size, relative to its parent's coordinate system. The location indicates the hot spot (or sensitive point) of the image. However, the image can be drawn in several different positions relative to this hot spot (see property *Position* in the table).

The element is interactive (if the parent is interactive) and triggers the method indicated by the corresponding action property when it is pressed, dragged or released.

Table of Properties		
Name	Description	Possible values
Image	The gif file with the image to be drawn	Constant: The name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable of type <i>String</i>
X	The X coordinate of the location of the image	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Y	The Y coordinate of the location of the image	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Z	The Z coordinate of the location of the image	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size X	The X component of the size of the image	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Y	The Y component of the size of the image	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Z	The Z component of the size of	Constant: Any constant number

	the image	Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Enabled	Whether the element is responsive to user interaction	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Scale X	A scale factor in the X axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Scale Y	A scale factor in the Y axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Scale Z	A scale factor in the Z axis to apply before drawing the element	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Position	The position of the shape relative to the hot spot	Constant: One of <i>CENTERED</i> , <i>HOR_CENTERED</i> , <i>VER_CENTERED</i> , <i>LOWER_LEFT</i> , <i>UPPER_LEFT</i> Variable: A variable of type <i>int</i>
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action

Text

Icon : 

Caption: An interactive text

Description: A *Text* is a drawable element that draws a string at a given location of its parent.

text

The string is drawn at the given location, using the specified font (which determines its size).


The location indicates the hot spot (or sensitive point) of the text. However, the text can be drawn in several different positions relative to this hot spot (see property *Position* in the table).

The element is interactive (if the parent is interactive) and triggers the method indicated by the corresponding action property when it is pressed, dragged or released.

Table of Properties		
Name	Description	Possible values
Text	The string to be drawn	Constant: Any string (of reasonable length) Variable: A variable of type <i>String</i>
X	The X coordinate of the location of the text	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Y	The Y coordinate of the location of the text	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Z	The Z coordinate of the location of the text	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Enabled	Whether the element is responsive to user interaction	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Font	The font used to display the text	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain</i> , <i>bold</i> , <i>italic</i> , <i>bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the

		class <i>java.awt.Font</i>
Position	The position of the text relative to the hot spot	Constant: One of <i>CENTERED</i> , <i>HOR_CENTERED</i> , <i>VER_CENTERED</i> , <i>LOWER_LEFT</i> , <i>UPPER_LEFT</i> Variable: A variable of type <i>int</i>
Color	The color used to display the text	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action

Trace

Icon : 

Caption: A sequence of points

Description: A *Trace* is a drawable element that visualizes a sequence of points in its parent.



Two datasets with no markers and connected

The points are drawn at the time they are added, one at a time, as marks at the given location, using the specified marker properties. They can also be connected by a segment, according to the *Connected* property.

The data set can be instructed to draw a maximum number of points. If so, a new point will cause the first one in the set to be discarded, thus acting as a strip chart recorder. If the *No Repeat* property is set to true and the point to be added equals the last one, the new point is ignored. This is useful when the parent is in autoscale state and we don't want static data to modify the scales of the parent.

Finally the trace can also be instructed to ignore a sequence of points before actually drawing a new one. This is useful if the number of points produced and sent to the data set is too large and we want to display a subset of them.

The element is not interactive and triggers no action.

Table of Properties		
Name	Description	Possible values
Points	The maximum number of points to draw	Constant: Any constant integer number. <i>0</i> means the sequence is infinite Variable: A variable of type <i>int</i>
Skip	The number of points to ignore before actually plotting one	Constant: Any constant integer number. <i>0</i> means that all points are drawn Variable: A variable of type <i>int</i> .
X	The X coordinate for the next point	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>

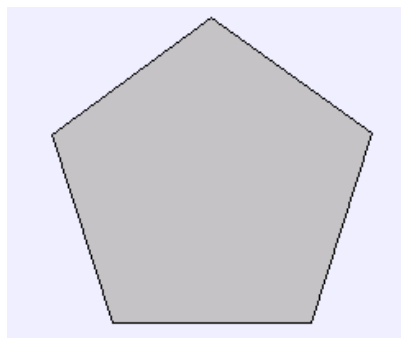
Y	The Y coordinate for the next point	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Z	The Z coordinate for the next point	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
No Repeat	Whether to ignore a point which is equal to the last one	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Connected	Whether to connect the markers. This affects only the next point to be added. Thus, changing this property dynamically can produce discontinuous curves	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Color	The color for the connecting lines	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Marker Shape	The shape for the markers	Constant: Either <i>NO_MARKER, CIRCLE, FILLED_CIRCLE, SQUARE</i> or <i>FILLED_SQUARE</i> Variable: A variable of type <i>int</i>
Marker Size	The size for the markers (in pixels)	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Marker Color	The color for the markers	See <i>Color</i> above

Poligon

Icon : 

Caption: A filled poligon

Description: This drawable corresponds to a closed poligon specified by a set of vertex points (x,y,z). The poligon may be drawn filled of hollow. The color for the border and the inside of the poligon may be specified.



The poligon is not interactive, hence it cannot modify the data for the vertex, nor trigger any action.

Table of Properties		
Name	Description	Possible values
X	The X coordinates of the vertex points	Constant: A <i>double</i> constant, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Y	The Y coordinates of the vertex points	Constant: A <i>double</i> constant, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Z	The Z coordinates of the vertex points	Constant: A <i>double</i> constant, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Connected	Whether each point is connected to the next one	Constant: Not applicable. By default all points are connected to the next one Variable: A variable array of type <i>boolean</i> . Each element of the array (except the last) indicates if the point must be connected to the next one (see also <i>Closed</i>)
Visible	Whether the element is	Constant: Either <i>true</i> or <i>false</i> .

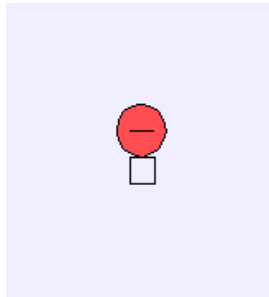
	visible	Variable: A variable of type <i>boolean</i>
Points	The number of vertex	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Closed	Whether the last point is connected to the first one	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Line Color	The color used to draw the border of the poligon	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> .The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Fill Color	The color used to fill the inside of the poligon. If unspecified, the poligon is not filled	See Line Color above

LightBulb

Icon : 

Caption: A variable color light

Description: This drawable corresponds to a light bulb which displays a light of a given color. The intensity of the light is associated to an integer value ranging from 0 (transparent) to 255 (opaque). Hence, modifying the value of the internal variable turns the light gradually on or off.



The light is not interactive, hence it cannot modify the intensity variable, nor trigger any action.

Table of Properties		
Name	Description	Possible values
X	The X coordinate of the center of the base of the light	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Y	The Y coordinate of the center of the base of the light	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Z	The Z coordinate of the center of the base of the light	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Radius	The radius of the bulb	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Intensity	The intensity (degree of transparency) of the light. 0 is transparent, 255 is opaque	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Color	The color used to draw the light	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> .

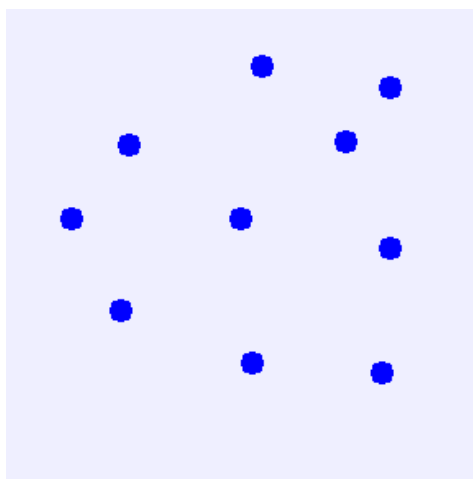
		Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> .The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Line Color	The color used to draw the lamp	See <i>Color</i> above

ParticleSystem

Icon : 

Caption: A set of particles

Description: A *ParticleSystem* is a set of several *Particle* elements.



If you understand how a *Particle* works, then you know how a *ParticleSystem* works. The only difference is that, for some properties, you will need to specify a whole array of values instead of a simple one. If, still, you specify a constant value, this value will apply for all the individual particles.


Table of Properties		
Name	Description	Possible values
X	The X coordinates of the locations of the particles	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Y	The Y coordinates of the locations of the particles	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Z	The Z coordinates of the locations of the particles	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Size X	The X components of the sizes of the particles	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points

Size Y	The Y components of the sizes of the particles	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Size Z	The Z components of the sizes of the particles	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Visible	Whether each individual element is visible	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Enabled	Whether each individual element is responsive to user interaction	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Elements	The number of individual elements in the set	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Scale X	A scale factor in the X axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Scale Y	A scale factor in the Y axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Scale Z	A scale factor in the Z axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Style	The type of shapes to draw (the same for all elements)	Constant: Either a simple dot or a filled or hollow circle or square. The valid values are <i>NO_MARKER</i> , <i>CIRCLE</i> , <i>FILLED_CIRCLE</i> , <i>SQUARE</i> , <i>FILLED_SQUARE</i> Variable: A variable of type <i>int</i>
Position	The position of the shapes relative to the hot spot (the same for all elements)	Constant: One of <i>CENTERED</i> , <i>HOR_CENTERED</i> , <i>VER_CENTERED</i> , <i>LOWER_LEFT</i> , <i>UPPER_LEFT</i> Variable: A variable of type <i>int</i>
Color	The color used to draw each individual element	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> ,

		<p><i>magenta, orange, pink, red, white, yellow.</i></p> <p>Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i>. The default is decided by your system</p> <p>Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i> applies to all elements. An array of <i>Object</i> variables of the class <i>java.awt.Color</i> specifies a color for each individual element</p>
On Press	The action to trigger when the mouse button is pressed on the element	<p>Constant: not applicable</p> <p>Variable: An action</p>
On Drag	The action to trigger when the mouse button is dragged on the element	<p>Constant: not applicable</p> <p>Variable: An action</p>
On Release	The action to trigger when the mouse button is released on the element	<p>Constant: not applicable</p> <p>Variable: An action</p>

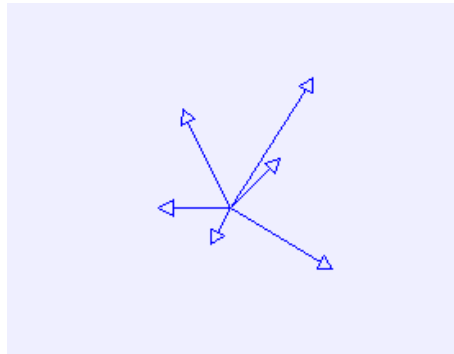
ArrowSet

Description: A

Icon : 

Caption: A set of vectors

Description: An *ArrowSet* is a set of several *Arrow* drawable elements.



If you understand how an *Arrow* works, then you know how an *ArrowSet* works. The only difference is that, for some properties, you will need to specify a whole array of values instead of a simple one. If, still, you specify a constant value, this value will apply for all the individual arrows.

Table of Properties		
Name	Description	Possible values
X	The X coordinates of the origins of the vectors	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Y	The Y coordinates of the origins of the vectors	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Z	The Z coordinates of the origins of the vectors	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Size X	The X components of the sizes of the vectors	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Size Y	The Y components of the	Constant: A constant number, meaning

	sizes of the vectors	the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Size Z	The Z components of the sizes of the vectors	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Visible	Whether each individual element is visible	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Enabled	Whether each individual element is responsive to user interaction	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Elements	The number of individual elements in the set	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Scale X	A scale factor in the X axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Scale Y	A scale factor in the Y axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Scale Z	A scale factor in the Z axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Style	The type of vectors to draw (the same for all elements)	Constant: Either an arrow, a segment of a segment with a square handle at the end point can be drawn. The valid values are <i>ARROW</i> , <i>SEGMENT</i> and <i>BOX</i> Variable: A variable of type <i>int</i>
Color	The color used to draw each individual element	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for

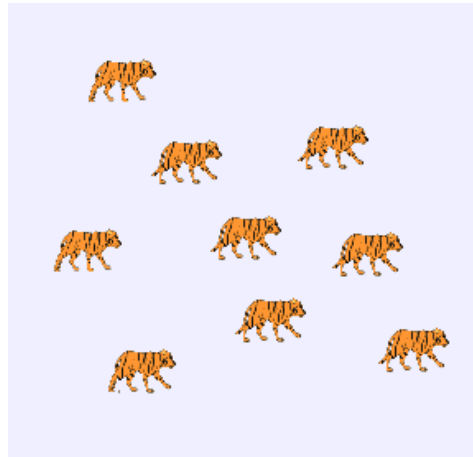
		instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i> applies to all elements. An array of <i>Object</i> variables of the class <i>java.awt.Color</i> specifies a color for each individual element
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action

ImageSet

Icon : 

Caption: A set of images

Description: An *ImageSet* is a set of several *Image* drawable elements.



If you understand how a *Image* works, then you know how a *ImageSet* works. The only difference is that, for some properties, you will need to specify a whole array of values instead of a simple one. If, still, you specify a constant value, this value will apply for all the individual images.

Table of Properties		
Name	Description	Possible values
Image	The gif file or array of files with the image or images to be drawn	Constant: A constant string (delimited by quotes or inverted commas), meaning the same image for all. The string must hold the name of an existing gif file. The file or URL location must be specified either as an absolute path or relative to Ejs' working directory Variable: A variable array of type <i>String</i> . Alternatively, a single <i>String</i> sets the same image for all the elements
X	The X coordinates of the locations of the images	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Y	The Y coordinates of the locations of the images	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points

Z	The Z coordinates of the locations of the images	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Size X	The X components of the sizes of the images	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Size Y	The Y components of the sizes of the images	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Size Z	The Z components of the sizes of the images	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Visible	Whether each individual element is visible	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Enabled	Whether each individual element is responsive to user interaction	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Elements	The number of individual elements in the set	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Scale X	A scale factor in the X axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Scale Y	A scale factor in the Y axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Scale Z	A scale factor in the Z axis to apply before drawing the individual elements	Constant: A constant number, meaning the same value for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same value for all points
Position	The position of the shapes relative to the hot spot (the same for all elements)	Constant: One of <i>CENTERED</i> , <i>HOR_CENTERED</i> , <i>VER_CENTERED</i> , <i>LOWER_LEFT</i> , <i>UPPER_LEFT</i>

		Variable: A variable of type <i>int</i>
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action

TextSet

Icon : 

Caption: A set of texts

Description: A *TextSet* is a set of several *Text* drawable elements.



If you understand how a *Text* element works, then you know how a *TextSet* works. The only difference is that, for some properties, you will need to specify a whole array of values instead of a simple one. If, still, you specify a constant value, this value will apply for all the individual texts.

Table of Properties		
Name	Description	Possible values
Texts	The string or array of strings to display	Constant: A constant string (delimited by quotes or inverted commas), meaning the same text for all Variable: A variable array of type <i>String</i> . Alternatively, a single <i>String</i> sets the same text for all the elements
X	The X coordinates of the locations of the texts	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Y	The Y coordinates of the locations of the texts	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Z	The Z coordinates of the locations of the texts	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Visible	Whether each individual element is visible	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type

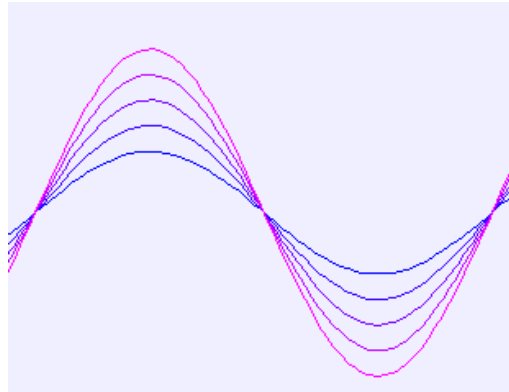
		<i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Enabled	Whether each individual element is responsive to user interaction	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Font	The font font used to display the texts (the same for all elements)	Constant: The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain</i> , <i>bold</i> , <i>italic</i> , <i>bold italic</i> . Example: <i>Monospaced,italic,18</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Font</i>
Elements	The number of individual elements in the set	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Position	The position of the texts relative to the hot spot (the same for all elements)	Constant: One of <i>CENTERED</i> , <i>HOR_CENTERED</i> , <i>VER_CENTERED</i> , <i>LOWER_LEFT</i> , <i>UPPER_LEFT</i> Variable: A variable of type <i>int</i>
Color	The color used to draw each individual element	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> .The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i> applies to all elements. An array of <i>Object</i> variables of the class <i>java.awt.Color</i> specifies a color for each individual element
On Press	The action to trigger when the mouse button is pressed on the element	Constant: not applicable Variable: An action
On Drag	The action to trigger when the mouse button is dragged on the element	Constant: not applicable Variable: An action
On Release	The action to trigger when the mouse button is released on the element	Constant: not applicable Variable: An action

TraceSet

Icon : 

Caption: A set of traces

Description: A *TraceSet* is a set of several *Trace* drawable elements.



If you understand how a *Trace* element works, then you know how a *TraceSet* works. The only difference is that, for some properties, you will need to specify a whole array of values instead of a simple one. If, still, you specify a constant value, this value will apply for all the individual traces.

Table of Properties		
Name	Description	Possible values
Points	The maximum number of points to draw for each element	Constant: A constant integer, meaning the same value for all Variable: A variable array of type <i>int</i> . Alternatively, a single <i>int</i> sets the same value for all the elements
Skip	The number of points to ignore before actually plotting one for each element	Constant: A constant integer, meaning the same value for all Variable: A variable array of type <i>int</i> . Alternatively, a single <i>int</i> sets the same value for all the elements
X	The X coordinates for the next set of points	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Y	The Y coordinates for the next set of points	Constant: A constant number, meaning the same coordinate for all Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Z	The Z coordinates for the next set of points	Constant: A constant number, meaning the same coordinate for all

		Variable: A variable array of type <i>double</i> . Alternatively, a single double sets the same coordinate for all points
Visible	Whether each individual element is visible	Constant: Either <i>true</i> or <i>false</i> , which applies to all the elements Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Elements	The number of individual elements in the set	Constant: Any constant integer number Variable: A variable of type <i>int</i>
No Repeat	Whether to ignore a point which is equal to the last one (the same for all the elements)	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Connected	Whether to connect the markers in each element. This affects only the next point to be added. Thus, changing this property dynamically can produce discontinuous curves	Constant: either <i>true</i> or <i>false</i> , meaning the same coordinate for all Variable: A variable array of type <i>boolean</i> . Alternatively, a single boolean sets the same value for all points
Color	The color for the connecting lines for each element	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system. This constant color will apply to all the element Variable: A variable array of type <i>Object</i> with elements of the class <i>java.awt.Color</i> . Alternatively, a single <i>Object</i> variable of the class <i>java.awt.Color</i> will set the color for all the elements
Marker Shape	The shape for the markers (the same for all the elements)	Constant: Either <i>NO_MARKER, CIRCLE, FILLED_CIRCLE, SQUARE</i> or <i>FILLED_SQUARE</i> Variable: A variable of type <i>int</i>
Marker Size	The size for the markers in pixels (the same for all the elements)	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Marker Color	The color for the markers of each element	See <i>Color</i> above

Surface

Icon : 

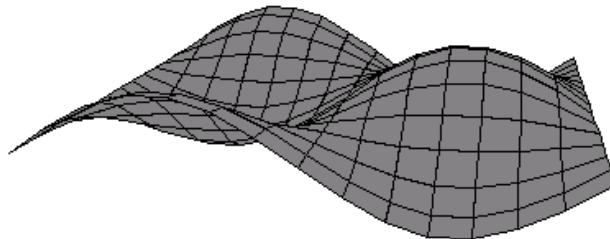
Caption: A 3D surface

Description: This drawable corresponds to a three dimensional surface of the form $(x,y,z) = (x(u,v), y(u,v), z(u,v))$. The surface may be drawn filled or in wire-frame mode. The color for the lines and the inside of the frames may be specified.

The data must be specified as a three-dimensional array. For instance the code

```
for (int i=0; i<n; i++) {
  for (int j=0; j<m; j++) {
    data[i][j][0] = -3.14 + (6.28*i)/(n-1);
    data[i][j][1] = -3.14 + (6.28*j)/(m-1);
    data[i][j][2] = Math.sin(data[i][j][0])*Math.cos(data[i][j][1]);
  }
}
```

where n and m equal 15 and $data$ has been declared with a dimension of $[n][m][3]$, produces the following surface (displayed in a *DrawingPanel3D*):




The surface is not interactive, hence it cannot modify its data, nor trigger any action.

Table of Properties		
Name	Description	Possible values
Data	The three dimensional array with the data. The last dimension must be 3, providing for each point the x,y , and z coordinates (see the example above)	Constant: Not applicable. Variable: A variable 3D array of type <i>double</i> . The array must be dimensioned like $[nu][nv][3]$, where nu is the number of u points for which a point in the surface is computed. Similarly, nv is the number of v points. The 3 doubles in the last index hold the values for the x, y and z coordinates of the point, respectively
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Close Bottom	Whether the first row of points should form a closed	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>

	poligon	
Close Top	Whether the last row of points should form a closed poligon	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Close Left	Whether the first column of points should form a closed poligon	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Close Right	Whether the last column of points should form a closed poligon	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Line Color	The color used to draw the lines of the surface. If unspecified the lines are not drawn	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Fill Color	The color used to fill the inside of the surface frames. If unspecified, the surface is drawn in wire-frame mode	See Line Color above

VectorField

Icon : 

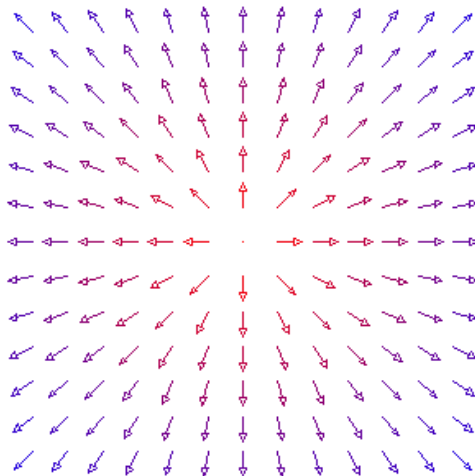
Caption: A field of vectors

Description: This drawable corresponds to a two- or three-dimensional set of vectors which can be used to display a field. The set of vectors are specified by giving its position in the plane or in the space, its size and an extra magnitude which is translated into a color code. The color of each vector is extrapolated linearly from a minimum color which corresponds to the minimum possible value of the magnitude, to a maximum color, corresponding to the maximum possible value of the magnitude.

The data must be specified as a three-dimensional array, for 2D vector fields, or as a four-dimensional array, for 3D vector field. For instance the code

```
for (int i=0; i<n; i++) {
  for (int j=0; j<m; j++) {
    double x = -1.0 + (2.0*i)/(n-1);
    double y = -1.0 + (2.0*j)/(m-1);
    double r = Math.sqrt(x*x+y*y);
    data[i][j][0] = x;
    data[i][j][1] = y;
    if (r>1.0e-8) {
      data[i][j][2] = x/r;
      data[i][j][3] = y/r;
      data[i][j][4] = r;
    }
  }
}
```

where n and m equal 15 and $data$ has been declared with a dimension of $[n][m][5]$, produces the following field (displayed in a 2D *DrawingPanel*):



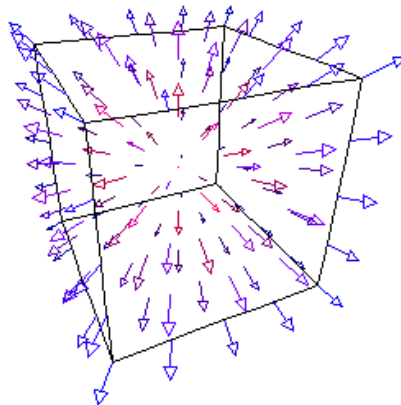
Similarly, the code

```

for (int i=0; i<n; i++) {
  for (int j=0; j<m; j++) {
    for (int k=0; k<p; k++) {
      double x = -1.0 + (2.0*i)/(n-1);
      double y = -1.0 + (2.0*j)/(m-1);
      double z = -1.0 + (2.0*k)/(p-1);
      double r = Math.sqrt(x*x+y*y+z*z);
      data2[i][j][k][0] = x;
      data2[i][j][k][1] = y;
      data2[i][j][k][2] = z;
      if (r>1.0e-8) {
        data2[i][j][k][3] = x/r;
        data2[i][j][k][4] = y/r;
        data2[i][j][k][5] = z/r;
        data2[i][j][k][6] = r;
      }
    }
  }
}

```

where n , m and p equal 5 and $data2$ has been declared with a dimension of $[n][m][p][7]$, produces the following field (displayed in a 3D *DrawingPanel*):



The vector field is not interactive, hence it cannot modify its data, nor trigger any action.

Table of Properties		
Name	Description	Possible values
Data	The three dimensional array with the data (see the examples above)	Constant: Not applicable. Variable: A variable 3D or 4D array of type <i>double</i>
Minimum	The minimum value of the magnitude that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum	The maximum value of the magnitude that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Levels	The number of different colors to distinguish	Constant: Any constant integer number Variable: A variable of type <i>int</i>

	between the minimum and the maximum	
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale	Whether the automatically get the extrema for the magnitude out of the data	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Min Color	The color that corresponds to the minimum value of the magnitude	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Min Color	The color that corresponds to the maximum value of the magnitude	See <i>Min Color</i> above
Zoom	A magnifying factor to apply before drawing the vectors	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>

Lattice

Icon : 

Caption: A visualization of a set of 0's and 1's

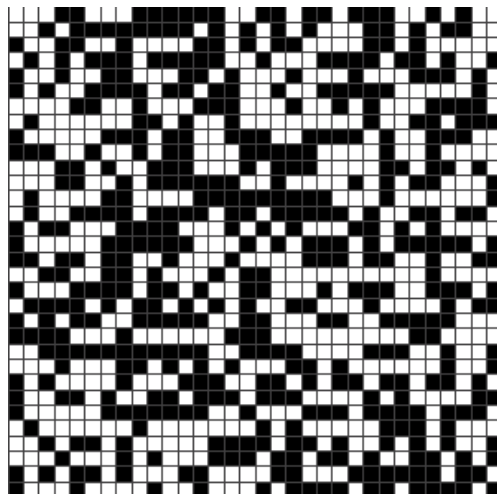
Description: A *Lattice* is a drawable element that displays a set of rectangles with one of two possible colors, depending on the value of the corresponding element of an array of integers.

Since no values are specified for the x and y components of the rectangles, it is best to display this drawable in a drawing panel which has the *Autoscale X* and *Autoscale Y* properties set to *true*.

The data must be specified as a two-dimensional array of integers. For instance the code

```
for (int i=0; i<n; i++)
  for (int j=0; j<n; j++)
    if (Math.random()<0.5) data[i][j] = 0;
    else data[i][j] = 1;
```

where n equals 32 and *data* has been declared of type *int* and with a dimension of $[n][n]$, produces the following field (displayed in a *DrawingPanel* with autoscales set to true and the extrema properties left empty),



The element is not interactive and triggers no action.

Table of Properties		
Name	Description	Possible values
Data	The data array with the values for the points	Constant: Not applicable. Variable: A variable 2D array of type <i>int</i> . The array must be dimensioned $[nx][ny]$ where nx and ny are the number of divisions in the x and y axes,

		respectively
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Alive Color	The color used to draw a rectangle if the corresponding value is 1 (in fact, if it is non zero)	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is <i>white</i> Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Dead Color	The color used to draw a rectangle if the corresponding value is 0	See <i>Live Color</i> above. The default is <i>black</i>

CheckerField

Icon : 

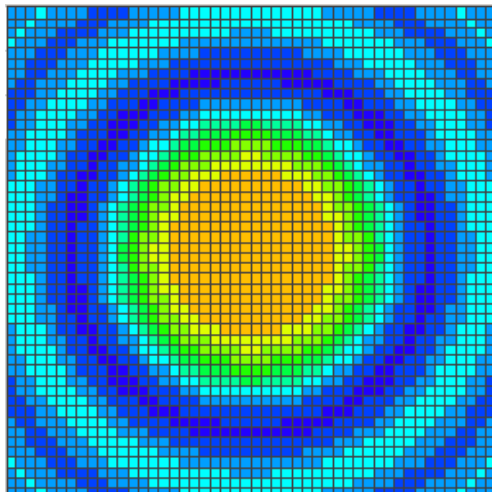
Caption: A checker visualization of a scalar field

Description: A *CheckerField* is a drawable element that displays color-coded rectangles with the same scalar value.

The data must be specified as a three-dimensional array of doubles. The last dimension is 3 and holds the x , y and z values for each point. The x and y values are used to locate the rectangles in the parent *DrawingPanel*'s area. The z value is considered a magnitude that must be extrapolated into a color code. For instance the code,

```
for (int i=0; i<n; i++) {
  for (int j=0; j<n; j++) {
    double x = -4.0 + i*8.0/(n-1);
    double y = -4.0 + j*8.0/(n-1);
    data[i][j][0] = x;
    data[i][j][1] = y;
    double p = (x*x + y*y)/2.0;
    if (p<1.0e-4) data[i][j][2] = 0.5;
    else data[i][j][2] = 0.5*Math.sin(p)/p;
  }
}
```

where n equals 48 and *data* has been declared of type *double* and with a dimension of $[n][n][3]$, produces the following field (displayed in a *DrawingPanel*),



The element is not interactive and triggers no action.

Table of Properties		
Name	Description	Possible values
Data	The three dimensional array	Constant: Not applicable.

	with the data. The last dimension must be 3, providing for each point the x, y , and z coordinates (see the example above)	Variable: A variable 3D array of type <i>double</i> . The array must be dimensioned like $[nx][ny][3]$, where nx is the number of x points for which the scalar value is computed. Similarly, ny is the number of y points. The 3 doubles in the last index hold the values for the x, y and z coordinates of the point, respectively. The z coordinate is used to compute the lines and to select the colors
Minimum Z	The minimum Z value that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum Z	The maximum Z value that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Levels	The number of lines to draw between the minimum and the maximum	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale Z	Whether to automatically compute the minimum and maximum values for Z	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Color Mode	The coding system for the colors	Constant: Either <i>spectrum</i> , <i>grayscale</i> , <i>dualshade</i> or <i>binary</i> Variable: A variable of type <i>int</i>

Contour

Icon : 

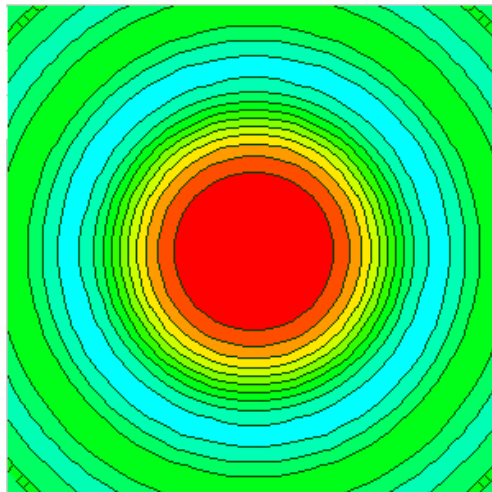
Caption: A contour display of a scalar field

Description: A *Contour* is a drawable element that displays lines connecting points with the same scalar value using a color-coded system.

The data must be specified as a three-dimensional array of doubles. The last dimension is 3 and holds the x , y and z values for each point. The x and y values are used to locate the rectangles in the parent *DrawingPanel*'s area. The z value is considered a magnitude that must be extrapolated into a color code. For instance the code,

```
for (int i=0; i<n; i++) {
  for (int j=0; j<n; j++) {
    double x = -4.0 + i*8.0/(n-1);
    double y = -4.0 + j*8.0/(n-1);
    data[i][j][0] = x;
    data[i][j][1] = y;
    double p = (x*x + y*y)/2.0;
    if (p<1.0e-4) data[i][j][2] = 0.5;
    else data[i][j][2] = 0.5*Math.sin(p)/p;
  }
}
```

where n equals 32 and *data* has been declared of type *double* and with a dimension of $[n][n][3]$, produces the following field (displayed in a *DrawingPanel*),



The element is not interactive and triggers no action.

Table of Properties		
Name	Description	Possible values
Data	The data array with the value	Constant: Not applicable.

	for the points	Variable: A variable 3D array of type <i>double</i> . The array must be dimensioned like $[nx][ny][3]$, where <i>nx</i> is the number of x points for which the scalar value is computed. Similarly, <i>ny</i> is the number of y points. The 3 doubles in the last index hold the values for the x, y and z coordinates of the point, respectively. The z coordinate is used to compute the lines and to select the colors
Minimum Z	The minimum Z value that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum Z	The maximum Z value that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Levels	The number of lines to draw between the minimum and the maximum	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale Z	Whether to automatically compute the minimum and maximum values for Z	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Color Mode	The coding system for the colors	Constant: Either <i>spectrum</i> , <i>grayscale</i> or <i>dualshade</i> Variable: A variable of type <i>int</i>

SurfacePlot

Icon : 

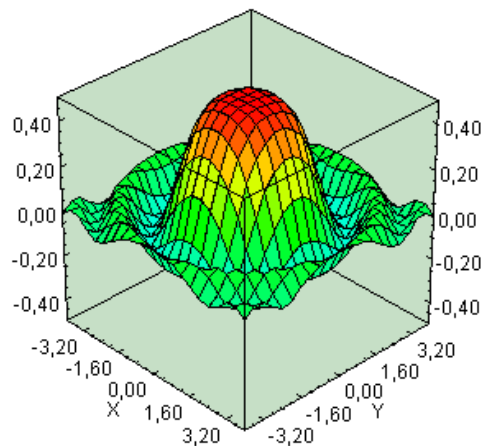
Caption: A 3D visualization of a scalar field

Description: A *Surface* is a drawable element that displays a 3D surface of the form $z=f(x,y)$, using a color-coded system.

The data must be specified as a three-dimensional array of doubles. The last dimension is 3 and holds the x , y and z values for each point. The x and y values are used to locate the rectangles in the parent *DrawingPanel*'s area. The z value is considered a magnitude that must be extrapolated into a color code. For instance the code,

```
for (int i=0; i<n; i++) {
  for (int j=0; j<n; j++) {
    double x = -4.0 + i*8.0/(n-1);
    double y = -4.0 + j*8.0/(n-1);
    data[i][j][0] = x;
    data[i][j][1] = y;
    double p = (x*x + y*y)/2.0;
    if (p<1.0e-4) data[i][j][2] = 0.5;
    else data[i][j][2] = 0.5*Math.sin(p)/p;
  }
}
```

where n equals 64 and *data* has been declared of type *double* and with a dimension of $[n][n][3]$, produces the following field (displayed in a *DrawingPanel*),



The element is not interactive and triggers no action.

Table of Properties		
Name	Description	Possible values

Data	The data array with the value for the points	Constant: Not applicable. Variable: A variable 3D array of type <i>double</i> . The array must be dimensioned like $[nx][ny][3]$, where nx is the number of x points for which the scalar value is computed. Similarly, ny is the number of y points. The 3 doubles in the last index hold the values for the x , y and z coordinates of the point, respectively. The z coordinate is used to compute the lines and to select the colors
Minimum Z	The minimum Z value that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Maximum Z	The maximum Z value that can be color-coded	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Autoscale Z	Whether to automatically compute the minimum and maximum values for Z	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Color Mode	The coding system for the colors	Constant: Either <i>spectrum</i> , <i>grayscale</i> or <i>dualshade</i> Variable: A variable of type <i>int</i>

Sphere

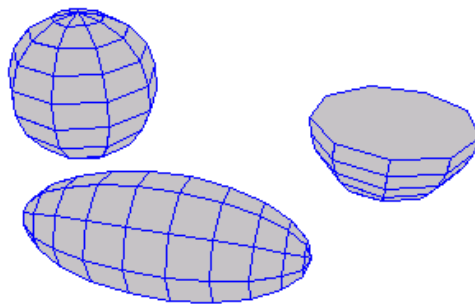
Icon : 

Caption: A 3D sphere (or ellipsoid)

Description: This drawable displays a three dimensional sphere. Actually, it displays an ellipsoid, or even part of it.

The ellipsoid is specified by giving the location of its center, the length of the semiaxes and its main direction. A particular choice for the direction allows using any vector as axes for the surface. This help produce slanted ellipsoids.

Finally, one can draw part of the ellipsoid by providing minimum and maximum value for the angles that describe the meridians and parallels of the ellipsoid.



Three sample sphere elements

The sphere is not interactive, hence it cannot modify its data, nor trigger any action.

Table of Properties		
Name	Description	Possible values
Center X	The x coordinate of the center of the ellipsoid	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Center Y	The y coordinate of the center of the ellipsoid	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Center Z	The z coordinate of the center of the ellipsoid	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Semiaxis X	The length of the first semiaxis (if direction is <i>x</i> , this corresponds to the <i>x</i> semiaxis)	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Semiaxis Y	The length of the second	Constant: Any constant number

	semiaxis (if direction is <i>x</i> , this corresponds to the <i>y</i> semiaxis)	Variable: A variable of type <i>int</i> or <i>double</i>
Semiaxis Z	The length of the third semiaxis (if direction is <i>x</i> , this corresponds to the <i>z</i> semiaxis)	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Direction	The direction in which to draw the ellipsoid	Constant: One of <i>x</i> , <i>y</i> , <i>z</i> or <i>custom</i> . <i>x</i> , <i>y</i> and <i>z</i> produce an ellipsoid with its first semiaxis parallel to the corresponding main axis and the two others in angles of 90 degrees from the first one. <i>custom</i> instructs the ellipsoid to use the <i>Axes</i> property to determine the direction of its axes Variable: A variable of type <i>int</i>
Axes	If <i>Direction</i> is set to <i>custom</i> , this must contain an array of 9 doubles with the coordinates of the three vectors to use as axes. The first three elements of the array form the first axis, and so on	Constant: Not applicable. Variable: A variable array of type <i>double</i> with dimension [9]
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Meridians	The number of divisions in the alpha angle of the ellipsoid	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Parallels	The number of divisions in the beta angle of the ellipsoid	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Min Alpha	The minimum value (in degrees) for the alpha angle of the ellipsoid. 0 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Max Alpha	The maximum value (in degrees) for the alpha angle of the ellipsoid. 360 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Min Beta	The minimum value (in degrees) for the beta angle of the ellipsoid. -90 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Max Beta	The maximum value (in degrees) for the beta angle of the ellipsoid. 90 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Close Bottom	Whether to close the bottom of the incomplete	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>

	ellipsoid whenever <i>Min Beta</i> is greater than <i>-90</i>	
Close Top	Whether to close the top of the incomplete ellipsoid whenever <i>Max Beta</i> is smaller than <i>90</i>	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Line Color	The color used to draw the lines of the ellipsoid. If unspecified the lines are not drawn	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Fill Color	The color used to fill the inside of the ellipsoid frames. If unspecified, the ellipsoid is drawn in wire-frame mode	See Line Color above

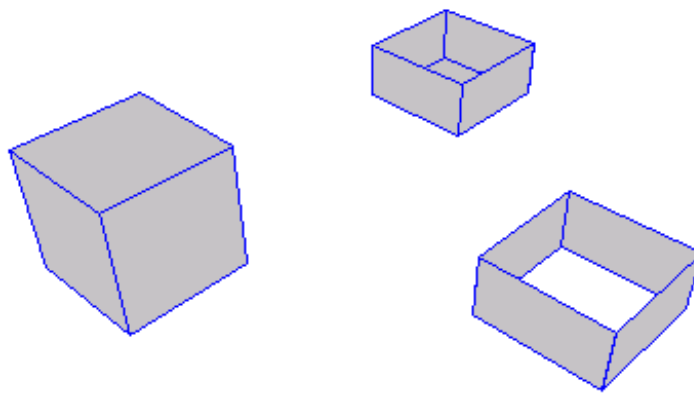
Cube

Icon : 

Caption: A 3D cube

Description: This drawable displays a straight three dimensional cube, with possible different sides length.

The cube is specified by giving the location of its origin and the length of its sides. One can also instruct the body whether to draw its top and bottom sides or not.



Three sample cube elements

The cube is not interactive, hence it cannot modify its data, nor trigger any action.

Table of Properties		
Name	Description	Possible values
Origin X	The x coordinate of the origin of the cube	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Origin Y	The y coordinate of the origin of the cube	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Origin Z	The z coordinate of the origin of the cube	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size X	The length of the cube along the <i>x</i> axis	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Y	The length of the cube along the <i>y</i> axis	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Size Z	The length of the cube	Constant: Any constant number

	along the z axis	Variable: A variable of type <i>int</i> or <i>double</i>
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Close Bottom	Whether to close the bottom of the cube	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Close Top	Whether to close the top of the cube	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Line Color	The color used to draw the lines of the cube. If unspecified the lines are not drawn	Constant: One of the following basic color names: <i>black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Fill Color	The color used to fill the sides of the cube. If unspecified, the cube is drawn in wire-frame mode	See Line Color above

Cilinder

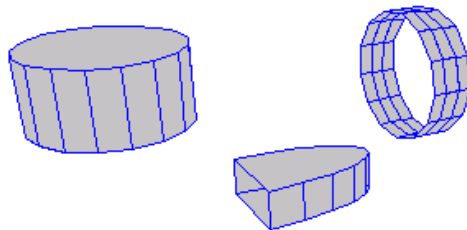
Icon : 

Caption: A 3D cilinder (with elliptical base)

Description: This drawable displays a three dimensional cilinder. Actually, it displays a cilinder with an elliptical base.

The cilinder is specified by giving the location of its center, the length of the semiaxes, its height and its main direction. A particular choice for the direction allows using any vector as axes for the cilinder. This help produce slanted cilinders.

One can instruct the cilinder whether to draw its top and bottom sides or not. Finally, one can draw part of the cilinder by providing minimum and maximum value for the angle that describes the circular parallels of the cilinder.



Three sample cilinder elements

The cilinder is not interactive, hence it cannot modify its data, nor trigger any action.

Table of Properties		
Name	Description	Possible values
Center X	The x coordinate of the center of the cilinder	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Center Y	The y coordinate of the center of the cilinder	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Center Z	The z coordinate of the center of the cilinder	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Semiaxis A	The length of the first semiaxis (if direction is <i>x</i> , this corresponds to the <i>x</i>)	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>

	semiaxis)	
Semiaxis B	The length of the second semiaxis (if direction is <i>x</i> , this corresponds to the <i>y</i> semiaxis)	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Height	The length of the side of the cilinder	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Direction	The direction in which to draw the cilinder	Constant: One of <i>x</i> , <i>y</i> , <i>z</i> or <i>custom</i> . <i>x</i> , <i>y</i> and <i>z</i> produce a cilinder with its first semiaxis parallel to the corresponding main axis and the two others in angles of 90 degrees from the first one. <i>custom</i> instructs the cilinder to use the <i>Axes</i> property to determine the direction of its axes Variable: A variable of type <i>int</i>
Axes	If <i>Direction</i> is set to <i>custom</i> , this must contain an array of 9 doubles with the coordinates of the three vectors to use as axes. The firt three elements of the array form the first axis, ans so on	Constant: Not applicable. Variable: A variable array of type <i>double</i> with dimension [9]
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Circle Sides	The number of divisions in the each parallel circle	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Height Sides	The number of divisions along the side	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Min Angle	The minimum value (in degrees) for the angle of the base. 0 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Max Angle	The maximum value (in degrees) for the angle of the base. 360 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Close Bottom	Whether to close the bottom of the cilinder	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Close Top	Whether to close the top of the cilinder	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Line Color	The color used to draw the lines of the cilinder. If unspecified the lines are not drawn	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to

		<i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>
Fill Color	The color used to fill the inside of the cilinder frames. If unspecified, the cilinder is drawn in wire-frame mode	See Line Color above

Cone

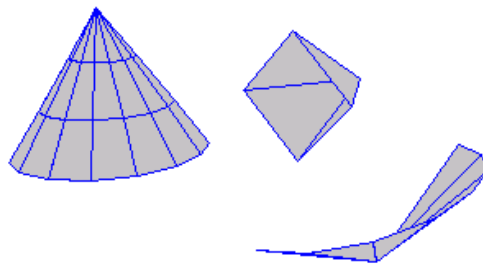
Icon : 

Caption: A 3D cone (with elliptical base)

Description: This drawable displays a three dimensional cone. Actually, it displays a cone with an elliptical base.

The cone is specified by giving the location of its center, the length of the semiaxes, its height and its main direction. A particular choice for the direction allows using any vector as axes for the cone. This help produce slanted cones.

One can instruct the cone whether to draw its bottom side or not. Finally, one can draw part of the cone by providing minimum and maximum value for the angle that describes the circular parallels of the cone.



Three sample cone elements

The cone is not interactive, hence it cannot modify its data, nor trigger any action.

Table of Properties		
Name	Description	Possible values
Center X	The x coordinate of the center of the cone	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Center Y	The y coordinate of the center of the cone	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Center Z	The z coordinate of the center of the cone	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Semiaxis A	The length of the first semiaxis (if direction is <i>x</i> , this corresponds to the <i>x</i> semiaxis)	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Semiaxis B	The length of the second	Constant: Any constant number

	semiaxis (if direction is <i>x</i> , this corresponds to the <i>y</i> semiaxis)	Variable: A variable of type <i>int</i> or <i>double</i>
Height	The length of the side of the cone	Constant: Any constant number Variable: A variable of type <i>int</i> or <i>double</i>
Direction	The direction in which to draw the cone	Constant: One of <i>x</i> , <i>y</i> , <i>z</i> or <i>custom</i> . <i>x</i> , <i>y</i> and <i>z</i> produce a cone with its first semiaxis parallel to the corresponding main axis and the two others in angles of 90 degrees from the first one. <i>custom</i> instructs the cone to use the <i>Axes</i> property to determine the direction of its axes Variable: A variable of type <i>int</i>
Axes	If <i>Direction</i> is set to <i>custom</i> , this must contain an array of 9 doubles with the coordinates of the three vectors to use as axes. The first three elements of the array form the first axis, and so on	Constant: Not applicable. Variable: A variable array of type <i>double</i> with dimension [9]
Visible	Whether the element is visible	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Circle Sides	The number of divisions in the each parallel circle	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Height Sides	The number of divisions along the side	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Min Angle	The minimum value (in degrees) for the angle of the base. 0 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Max Angle	The maximum value (in degrees) for the angle of the base. 360 by default	Constant: Any constant integer number Variable: A variable of type <i>int</i>
Close Bottom	Whether to close the bottom of the cone	Constant: Either <i>true</i> or <i>false</i> . Variable: A variable of type <i>boolean</i>
Line Color	The color used to draw the lines of the cone. If unspecified the lines are not drawn	Constant: One of the following basic color names: <i>black</i> , <i>blue</i> , <i>cyan</i> , <i>darkGray</i> , <i>gray</i> , <i>green</i> , <i>lightGray</i> , <i>magenta</i> , <i>orange</i> , <i>pink</i> , <i>red</i> , <i>white</i> , <i>yellow</i> . Alternatively, the red, green and blue integer components of the color, from 0 to 255, separated by commas. for instance, <i>0,0,255</i> is equivalent to <i>blue</i> . The default is decided by your system Variable: An <i>Object</i> variable of the class <i>java.awt.Color</i>

Fill Color	The color used to fill the inside of the cone frames. If unspecified, the cone is drawn in wire-frame mode	See Line Color above
------------	--	----------------------



E. Ejs advanced reference

This appendix describes further characteristics of **Ejs** that don't fit in the general manual. They are considered advanced features and are therefore reserved for the more skilled user.

Personalizing the list of view elements

Ejs can be run with a special command-line option to display less elements than are actually available. This can be useful to configure a panel for the view which is easier to cope with for newcomers. Instead of frightening the user with a long list of view elements to learn, you can select a handful of them, those that you and your users are more likely to need, and hide the others.

It can also be useful to, as more and more elements are added to **Ejs**, keep the number of elements offered at a time under reasonable limits.

Finally, it can be also useful to presenty different choices of view elements for different types of tasks. For instance, one could run **Ejs** with a set of view elements specially devoted for 3D graphics only, or for electric circuits only (when and if these elements are added to **Ejs**, of course ☺), and so on.

To do this, you need to modify the batch file with which you want to run **Ejs**. The best way is to copy the one you are using now³ and give the copy an appropriate name, for instance, *Ejs_simple.bat*. In it, you have to edit the last line and append, to the end of it, the following text

```
-elements simpleElements.txt
```

where *simpleElements.txt* (or any other name) must be a text file that you must create in your **Ejs data** directory⁴. The contents of this file must be similar to the following:

```
Containers=Frame Dialog Panel EMPTY DrawingPanel PlottingPanel  
DrawingPanel3D  
Basic=Label Button CheckBox Slider Field TextField Bar  
Drawables=Particle Arrow Image Text Trace EMPTY EMPTY ParticleSet  
ArrowSet ImageSet TextSet TraceSet
```

³ See section 2.2 of the manual.

⁴ See section 2.3 of the manual to locate this directory.

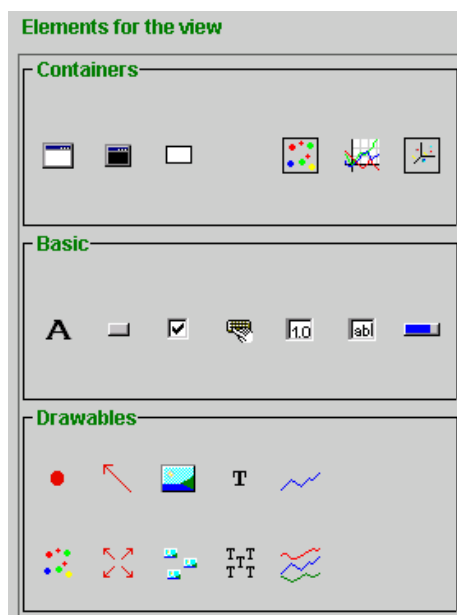
That is, it must contain one line for each of the entries *Containers*, *Basic* and *Drawables*, corresponding to the three group of possible view elements. For each group, you need to specify the list of elements that are offered, in one single line (although I have been forced to break some of the lines above because they do not fit in the page width of this document) and separated by blank spaces.

The names of the elements must match exactly any of the existing elements, as listed in appendix D. The special keyword EMPTY allows you to separate icons by leaving an empty space.

For instance, my Windows *Ejs_simple.bat* batch file reads

```
set JAVAROOT=c:\jdk1.3
set EjsDir=Simulations
%JAVAROOT%\bin\java -classpath
  %JAVAROOT%\lib\tools.jar;%JAVAROOT%\jre\lib\rt.jar;data\osejs.jar;data\
  HotEqn.jar -Dcodebase=. -Duser.home="%EjsDir%"
  org.colos.ejs.osejs.Osejs -locale es ES -elements simpleElements.txt
```

My *simpleElements.txt* file, which is in the *data* directory, reads exactly as the example above and, when I run *Ejs_simple.bat*, I get the following simplified view panel:



Running Ejs with different sets of options

As you know from section 4.1 of the manual, **Ejs** has some options that can be used to specify things like the location at start-up of **Ejs** main window, how will **Ejs** generate Html pages (if any) , and others...

This is most easily changed by using **Ejs** option dialog, as described there. However, you can also use a command line option similar to the previous one.

This can be of use if you want to prepare different sets of options to be used by different users or even by yourself under different circumstances.

For this, you'll need to edit the batch file with which you want to start **Ejs** and append to it the text

```
-options myOptions.txt
```

where *myOptions.txt* (or any other name) must be a text file that you must create in your **Ejs** *data* directory. The contents of this file must be similar to the following:

```
position=CENTER
generateHtml=ONE_PAGE
removeJavaFile=true
showHiddenPages=false
font=<default>
```

Again, the file is made of entries, each of them with an option. Each entry must be in a single, different line.

The accepted entries and options are listed in the table below.

Table of Configurration Options		
Entry	Description	Possible values
position	The start-up location of Ejs main window	One of <i>CENTER</i> , <i>TOPLEFT</i> or <i>CUSTOM:x,y</i> In this last case, <i>x</i> and <i>y</i> stand for the location on the screen (in pixels) of the upper-left corner of Ejs main Window
generateHtml	How to generate Html files	One of <i>LEFT_FRAME</i> , <i>TOP_FRAME</i> , <i>ONE_PAGE</i> or <i>NONE</i>
removeJavaFile	Whether to remove the generated Java file after running a simulation	Either <i>true</i> or <i>false</i>
showHiddenPages	Whether to show hidden pages	Either <i>true</i> or <i>false</i>
font	The default font	The family name, style and size of any font supported by the system, separated by commas. Style must be either: <i>plain</i> , <i>bold</i> , <i>italic</i> , <i>bold italic</i> . Example: <i>Monospaced,italic,18</i> . The special tag <i><default></i> uses the default font as decided by your system

This page intentionally left blank