

## 2 Scripting PADs II: Specifying objects and grading criteria

### 2.1 KinematicsPADs

#### 2.1.1 Overview

In order to understand how to write questions, and in particular grading criteria, it may help to have a brief overview of how the PADs work internally. The data in the KineticPADs (TablePAD, EquationPAD, GraphPAD, StrobePAD, MotionPAD and VideoPAD) is in the form of time, horizontal and vertical position. The data is stored internally in a triplet of time, horizontal (x) position and vertical (y) position, or in the form of  $(t, x, y)$ . In some applets, such as GraphPAD, some of those values are not used or visible, but is still present because the same data storage and grading engine is used for all of them. (In fact, the only real difference between the different KineticPADs is how data is displayed.) Every point on a graph or strobe diagram, every line in a table, or every frame or group of frames in a motion or video, is represented by an internal triplet. They are stored in sequential list by the  $t$  value. The KineticPAD grading engine calculates a fit to these points according to the fit specified in the ORDER parameter, producing a set of coefficients for the fit. These coefficients are used both internally (such as for drawing the line of fit in GraphPAD) and for evaluating the correctness of the response. If the ORDER parameter is 0 (point to point fit), the grading engine uses each sequential pair of points to calculate the y-intercept and slope of each line segment defined by the two points. If two or more segments defined this way have the same slope and  $y$  intercept, they will be counted internally as a single segment. This way students are not penalized for adding intermediate points to a straight line. Segments that are vertical are excluded from the list. If the ORDER parameter is greater than zero (polynomial fit), the grading engine performs a least-squares fit to the entire set of points to fit a polynomial. If the ORDER parameter is  $-1$  (power fit) the logarithm of the time and position values is taken and a linear function is fitted to the logarithm of the values.

Fit type	Point-to-Point fit	Polynomial fit	Power fit
ORDER	0	$n > 0$	-1
Equation	$x = c_{0sx} + c_{1sx}t \{t_n < t < t_{n+1}\}$ $y = c_{0sy} + c_{1sy}t \{t_n < t < t_{n+1}\}$ <sup>i</sup>	$x = c_{0x} + c_{1x}t + c_{2x}t^2 + \dots + c_{nx}t^n$ $y = c_{0y} + c_{1y}t + c_{2y}t^2 + \dots + c_{ny}t^n$	$x = c_{0x}t^{c_{1x}}$ $y = c_{0y}t^{c_{1y}}$

Example

Data	Point-to-Point fit	Polynomial fit ( <i>order=2</i> )	Power fit
(1.0, 3.0, 1.0)			
(2.0, 5.0, 2.5)	$x = 2t + 0, y = 3t + 0 \{0 < t < 1\}$	$x = 2t + 1$	$x = 2t^1$
(3.0, 7.0, 5.0)	$x = 2t + 0, y = 1t + 2 \{1 < t < 2\}$	$y = 0.5t^2 + 0t + 0.5$	$y = 0.92t^{1.59}$ <sup>ii</sup>
(4.0, 9.0, 8.5)	$x = 2t + 0, y = -1t + 6 \{2 < t < 3\}$	$\{1 < t < 5\}$	$\{1 < t < 5\}$
(5.0, 11, 13)	$x = 2t + 0, y = -3t + 12 \{3 < t < 4\}$		

#### 2.1.2 Specifying response and reference points

The form for importing and exporting data to and from the applet is a string of  $(t, x, y)$  triplets. The triplets are separated from each other by semicolons, and members of each triplet from each other by commas. Thus the data from the example above would be represented as:

1.0, 3.0, 1.0; 2.0, 5.0, 2.5; 3.0, 7.0, 5.0; 4.0, 9.0, 8.5; 5.0, 11, 13;

This is the format used to specify points in the KinematicPAD when it is originally loaded as the value of the RESPONSE parameter, to specify a reference line/object via the REFERENCEVALUE parameter, to insert values via the setResponse() method (e.g. from JavaScript), or retrieve the current status of the applet via the getResponse() function. Examples follow below.

<sup>i</sup> The numbering of segments  $n$  goes from 0 to number of segments  $-1$ . If three or more adjacent points are co-linear, the combined segment is counted as one segment. Vertical segments are ignored.

<sup>ii</sup> Note that this is not an exact fit, since the best curve through the points would not pass through  $(0,0)$ .

Initial load: `<Param name="Response" value="1.0, 3.0, 1.0; 2.0, 5.0, 2.5; 3.0, 7.0, 5.0;">`

Reference: `<Param name="ReferenceValues" value="1.0, 3.0, 1.0; 2.0, 5.0, 2.5; 3.0, 7.0, 5.0;">`

Setting: `document.PADname.setResponse("1.0, 3.0, 1.0; 2.0, 5.0, 2.5; 3.0, 7.0, 5.0;");`

Retrieve: `response = document.PADname.getResponse();` (Variable 'response' contains string).

### 2.1.3 Grading criteria

When the applet is requested to evaluate the current status via the `isCorrect()` or `computeFeedback()` parameters, it checks the status according to the criteria specified by the `Correct` parameter. The values available for checking against are the fit parameters, the beginning and end times, the values of points themselves, and mathematical combinations. The basic condition is that a value or set of values lie within a specified range. If the condition is met, an optional feedback message can be provided back to the user, and if the condition is not met a different optional feedback message can be provided. If all conditions specified by the `CORRECT` parameter are met, the applet is considered to be correct; if any are not met, it is considered to be incorrect.

The conditions for the applet to be correct are specified in the `CORRECT` parameter by one or more Criteria separated by semicolons. The basic form for a Criteria is:

`{modifier} value1 range {, correctMessage{, incorrectMessage}};`

The *value* may be a single number or a set of numbers and are discussed in the next section.

The *range* is specified by one or two values and the comparison operators, '=', '>', '<' and '+/-' to specify equal, greater and less than, and tolerance range. Examples:

<code>value1 = value2 +/- value3</code>	<code>value1 between value2-value3 and value2+value3</code>
<code>value1 &gt; value2 &lt; value3</code>	<code>value1 greater than value2 and less than value3</code>
<code>value1 &gt; value2</code>	<code>value1 greater than value2</code>
<code>value1 &lt; value2</code>	<code>value1 less than value2</code>

Order of greater than and less than signs does not matter. If there are duplicates, the last one is the one that matters. If the equal sign is used, a tolerance MUST be specified, otherwise the tolerance will sometimes be taken to be zero, and any number will fail. The applet will not perform any validity checking to detect an impossible range being specified, such as `value >3 <2`. (This can actually be useful at times in combinations with modifiers.)

If the *value* specified return multiple values, then multiple comparisons will be made. (This primarily results from reference values with the wild card character. See section 2.1.4.1) If *value1* contains multiple values and *value2* and *value3* contain only one, each value of *value1* will be compared to the same *value2* and *value3*. If *value1* contains a single value and *value2* and/or *value3* contain multiple values, *value1* will be compared to each range specified by *value2* and *value3*. If both *value1* and the range specified contain multiple values, the first will be compared to the first, the second to the second, and so forth. If one has fewer values than the other does, it will start over again with the first value, but it is not recommended to specify Criteria where there will be different numbers of multiple values.

The *modifier* is optional, and will either affect how the comparison is carried out or modify the final result. The first four describe how to handle multiple or missing values:

<code>allnomissing</code>	<code>value1</code> must contain at least one value and each value must satisfy the criteria.
<code>allmissing</code>	Each value <code>value1</code> contains must satisfy the criteria, but may be empty.
<code>onemissing</code>	<code>value1</code> must contain at least one value and exactly one satisfies the criteria.
<code>oneormissing</code>	<code>value1</code> is empty or exactly one value contained in it satisfies the criteria.
<code>nocheck</code>	Return correct without performing a check.
<code>not</code>	Return true if the criteria is not met, false if it is met.

The default behavior is `allnomissing`.

The *messages* are String Values, most commonly a string enclosed in quotes. If the value of *correctMessage* is given and it is not an empty string, it will be appended to the string supplied via the

computeFeedback() parameter if the criteria is satisfied. The same thing happens with *incorrectMessage* if the criteria is not satisfied.

## 2.1.4 Values

The numerical values that can be used in the Criteria consist of the fit coefficients, starting and end times, values of individual points, mathematical combinations of them, and actual numbers. A number may be supplied for any value, such as “2” or “-1.34” and the like.

### 2.1.4.1 References to internal values

References to internal data consist of values of individual points, starting and ending times of a segment or the entire set of data, and coefficients of the fit as described in the overview section.

Point $n$ , t/x/y	$pn.e$	$n$ is the number of the point (counting from zero) and $e$ is the element index: 0 for t, 1 for x, 2 for y. Example: p1.0 (time of second point), p4.2 (y of fifth point).
Time {of segment $n$ } start/end	$t\{sn\}.e$	$e$ is which time, 0 for the start of the graph or segment, 1 for the end of graph or segment. If ORDER=0, the form is $tsn.e$ where $n$ is the $n^{\text{th}}$ line segment (counting from zero). Examples: t.1 is the last time (power or polynomial graph), ts2.0 is the beginning time on the third segment of a line segment graph.
Coefficient of $p$ term {of segment $n$ } x/y	$cp\{sn\}.e$	$p$ is the power or order of the coefficient, $e$ is the element ( $x=0, y=1$ ), and if ORDER=0, $n$ is the number of the segment (counting from zero). Examples: c2.0 is the coefficient of the square term in the fit of $x(t)$ , c0s1.1 is the coefficient of the intercept ( $t^0$ ) term of the $y(t)$ for the second line segment.

**Table 1: Reference Values in KinematicPADs**

The wildcard character “#” can be used to replace one or more characters and all values that match will be returned. For example “c1s#.x” will return the slope for all the segments in a point-to-point graph. “c#” would return all the coefficients. The wildcard character may only appear once in an expression.

References to values in other KineticsPADs on the page can be made by appending “value.appletname.” onto the front of the reference, where *appletname* is the name of the applet containing the value desired. Example: “value.GraphPAD1.c1.0” would refer to the slope of the  $x(t)$  fit in the applet named “GraphPAD1.” The PAD *appletname* must be declared as a source. See the section on PAD communication for more information.

### 2.1.4.2 Mathematical values

Numbers, references and other mathematical values may be combined in a math value. The general form is:

$$\{function\}[value1 \textit{op} value2 \{op value3 \{op value4 \{...\}\}\}]$$

*function* is an optional function, such as sine or absolute value operating on the result of the expression in the square brackets. Each *value* is any valid value, whether a number, a reference value, or another mathematical expression. *op* is an operator. Valid operators are:

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Raise to a power

Example: “[c1.0 \* p1.0 + c0.0 – p1.1]” would calculate the difference of the  $x(t)$  fit at the time of the second point from that point’s  $x$  position. If wildcard references are used, *values* may be actually a list of values. If one of the values in an operation is single valued and the other multi-valued, the single-valued value will be multiplied/added/etc. to each value of the other, resulting in a multi-valued result. If both are

multi-valued, the first value will be combined with the first, the second with the second, and so forth. It is possible to combine multi-valued items with different numbers of values, but not recommended.

NOTE: the mathematical operations DO NOT RESPECT ORDER OF OPERATION. Instead they are simply evaluated left to right. “[1 \* 3 + 2]” would result in 5, but “[2 + 1 \* 3]” would give 9. If in doubt, nest math values inside each other, such as “[2 + [1 \* 3]]” in order to get 5.

abs	Take the absolute value.
acos	Compute the inverse cosine (in radians).
asin	Compute the inverse sine (in radians)..
atan	Compute the inverse tangent (in radians)..
average	Calculate the average of the multiple values calculated inside the expression. Returns a single value.
cos	Compute the cosine (in radians).
count	Count the number of values calculated inside the expression. Returns a single value.
difference	Return a set of values where each value is the difference between adjacent values calculated inside the expression. Return $n-1$ values.
exp	Calculate the exponential function of the expression ( $e^x$ ).
first	Return the first value in a set of values.
last	Return the last value in a set of values.
log	Calculate the natural logarithm of the expression.
math	No effect (same as nothing present).
max	Return the maximum value from a set of values.
mid	Return a set of values where each value is the midpoint between adjacent values calculated inside the expression. Return $n-1$ values.
min	Return the minimum value from a set of values.
round	Rounds values to the nearest integer.
sign	Returns 1 if the value is positive, 0 if it is zero, and -1 if negative.
sin	Compute the sine (in radians).
sqrt	Compute the square root of the expression.
sum	Returns the sum of all the values in the expression.
tan	Compute the tangent (in radians).

**Table 2: Mathematical functions in KinemeticPADs**

### 2.1.4.3 Concatenation value

The concatenation value can be used to provide a feedback message that depends on current values of the exercise, such as reporting current values of parameters or conditional messages (see next section). It obtains the string values from two or more values and concatenates them. It takes the form:

string[*value1* + *value2* {+ *value3* {+ *value4* {...}}}]

It is identified by ‘string’ at the beginning and can contain any combination of any type of Values.

### 2.1.4.4 On Value

The On Value provides a conditional value. Depending on the value of one Value, the On Value returns the value from a list. It takes the form:

on[*condition*, *value0*, *value1* {, *value2* {...}}]

The Value specified by *condition* is evaluated as a number. If it evaluates as zero or negative, the On Value returns the value of *value0*. If it evaluates to one (1), the On Value returns the value of *value1*, and so forth until the end of the list. If *condition* evaluates to a value greater than the index of the last value, the value of the last one in the list is returned. For example, if there are supposed to be two segments, the following feedback message could be provided:

on[count[c0s#0], ‘Draw a line’, ‘Not enough segments’, ‘Correct number’, ‘Too many segments’]

### 2.1.4.5 Useful examples

Here are some examples to show different uses of mathematical values that may be useful.

count[c0s#.0]	Number of segments in the PAD.
[c0.0*-1/c1.0]	Time at which the linear fit $x(t)$ crosses $x=0$ .
[max[p#.2]-min[p#.2]]	Range of y values present.
average[ mid[p#.1]*difference[p#.0]]	Area under the curve (numerical integration).
sum[sign[p#.2+abs[p#.2]]]	Number of y values greater than zero.

**Table 3: Examples of Values in KinematicPADs**

### 2.1.4.6 Note about PAD version 0.9

The way the correct values were specified changed between version 0.9 and 1.0 of the KineticPADs, so some of the examples on the website and in WebAssign use the old way. The distinguishing feature is the archive file; if the specified archive file ends with "\_0\_9.jar" that is a version 0.9 that uses a different method for specifying correct, tolerance, and what to do with missing or extra parameters than the archives that end with "\_1.jar". No other parameters changed. In most cases I recommend you ignore the old way and only use the new way, which is more powerful and flexible, and easier to understand.

## 2.2 VectorPAD

### 2.2.1 How it works

Almost all the data and parameters related to display and functionality are associated with individual vectors. This allows, for example, vectors and bars to be displayed on the same screen, and some to be unmovable while others can be drawn, moved and deleted. These different fields can be classified in several groups: primary, secondary, calculated, and functional. All the fields associated with a vector are summarized in Table 4 below.

Classification	Field	Type	Function
Primary	Name1	String	Labeling of vector
	Name2	String	Labeling of vector
	Name3	String	Labeling of vector
	StartX	number	Starting horizontal position of vector
	StartY	number	Starting vertical position of vector
	TransX	number	Horizontal displacement of vector
	TransY	number	Vertical displacement of vector
Secondary	StartUnit	number	Extra label for vector
	TransUnit	number	Extra label for vector
Calculated	StartD	number	Starting position of vector (angle)
	StartR	number	Starting position of vector (mag.)
	TransD	number	Displacement of vector (angle)
	TransR	number	Displacement of vector (mag.)
	CROSS <sup>iii</sup>	number	(StartX,StartY)x(TransX,TransY)
	EndX <sup>iii</sup>	number	StartX+TransX
	EndY <sup>iii</sup>	number	StartY+TransY
Display	Color	number	RGB Color, R*256 <sup>2</sup> +G*256+B
	Label	String	Label to display on screen
	LabelPosition	number	Where (relative to vector) to display label. 0: end, 1: start, 2: middle (with offset), 3:center (no offset)
	Size	number	Half-width of vector (in pixels).
	Style	String	"Vector" (or "0"), "HorizontalBar" (or "1"), "VerticalBar" (or "2"), "Segment" (or "3"), "OutOfPlane", "Rectangle" or name of *.gif file to be loaded (& declared with ICONLIST).
	TransScale	number	Multiplicative factor to scale displacement of vectors on screen; if set to 0.5, a displacement vector of 1 will appear half the length on the screen as the distance from origin to a start of magnitude 1.
	UseDegree	boolean	Whether to use degrees or radians in the display and calculation of values.
Function	Excluded <sup>iv</sup>	boolean	If true, the vector will not be interactive nor will be kept in the list available for evaluation.
	MayChangeName	boolean	If false, user can not change any name fields.
	MayChangeStart	boolean	If false, user can not change the start position.
	MayChangeTrans	boolean	If false, user can not change displacement.
	MayDelete	boolean	If false, user can not delete.

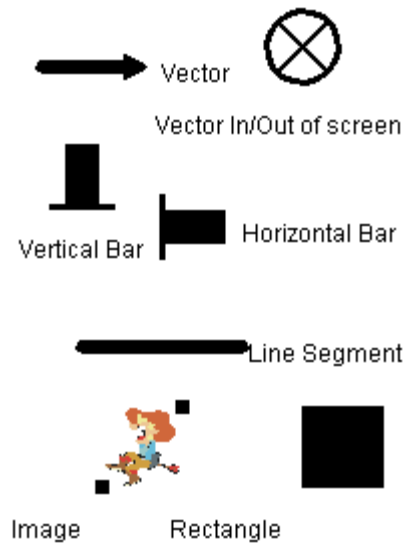
**Table 4 : Fields in VectorPAD**

The primary fields are those containing data about the name, position and displacement of a vector. Since these are the only fields that will always be provided by a `getResponse()` function, they are the only ones

<sup>iii</sup> May not be directly set.

<sup>iv</sup> May only be set in the RESPONSE parameter.

that are guaranteed to be saved and so they should be the only ones that should be evaluated for determining if work is correct or not. The secondary fields are additional space for labels (such as units for the numbers), but could be used for other things. The calculated fields include position and displacement and displacement in terms of angle and magnitude, as well as end position and the cross of the position with displacement. Some of the calculated fields are read-only. The rest of the fields have special properties with respect to display of the vector and functionality. There are fields to specify what label (if any) to display with the vector and where, color, width, and style: vector (with a single arrowhead), vertical and horizontal bars for bar charts, line segment with no arrow head, or loading a \*.gif image. See Figure 1. Function parameters can limit the user from re-labeling, or moving the vector, or cause it to be an entirely non-interactive, non-accessible background image. Data type is nominal in that all data in VectorPAD can be supplied in either string or numerical format; if a nominal number type is asked for a string value, it will produce a formatted string of the number. If a nominal string is asked for a number, it will attempt to convert the string to a number, returning zero if it fails. This is an important feature for the discussion of Values in the upcoming section. The fields are summarized in the chart below by classification, name, nominal type of data, and summary of its purpose/function.



**Figure 1: VectorPAD Object types**

### 2.2.2 Specifying response

Response data is set and retrieved by `setResponse()`, `getResponse` methods and the `RESPONSE` parameter through a string that consists of data for individual vectors separated by semicolons. The minimum string to specify a vector is

*name1, name2, name3, startX, startY, transX, transY;*

where *name1* through *name3* are strings of text in quotes, and the last four values are numerical. For example, one might specify a vector representing the force of gravity from the earth on a box as:

`'F(G)', 'earth', 'box', 0, 0, 0, -10;`

Additional fields may also be specified in value/key pair format, where the field name is set equal to the value. For example, to set the color and label we could use

`'F(G)', 'earth', 'box', 0, 0, 0, -10, color = 196 label="Weight of box";`

The primary fields may also be redefined in value/key pair format after the first part, which can sometimes be valuable to take care of the power of Values, discussed next.

### 2.2.3 Values

VectorPAD achieves a great flexibility through the use of Values. We have already seen the two most common values, the static number and the static string values in the above section. But values can be defined so that their value depends on other values. Fields of a vector can be set equal to a Value. For example, if one wanted to allow a student in the above example to change the type of force, it would make sense to make the label reflect that. In that case we could instead specify:

`'F(G)', 'earth', 'box', 0, 0, 0, -10, label=string[name1 + " of the "+name2+ " on the "+name3];`

which in this case would output "F(G) of the earth on the box", but if the student changed *name1* and *name2* to 'F(N)' and 'ground', the label would automatically read "F(N) of the ground on the box".

Values may be static, may depend on the value of other Values, or may depend on Criteria (discussed in Section 2.2.4. ). Note that, as mentioned above, all Values can provide either a string or numerical value, depending on the context, by attempting to translate one to the other.

### 2.2.3.1 Static values

Constant values have already been demonstrated above. A string constant value is specified by a string enclosed in quotes and a number constant consists of a string not enclosed in quotes that is a legitimate number (possible leading negative, digits 0-9, and a decimal point). If a value is specified that VectorPAD can not identify as another type, it will assume a string type.

### 2.2.3.2 Reference values

Reference Values refer to the current values in the fields of one or more vectors. The example above used reference values to refer to the contents of the fields 'name1', 'name2', and 'name3'. The full format for a Reference Value is

$fieldName[Criteria]$

where  $fieldName$  is the name of a valid field listed in Table 4 and the  $Criteria$  is a Criteria producing a list of vectors. Criteria will be discussed in Section 2.2.4, but for the purposes of this discussion, they will identify a set of vectors meeting a particular condition, and the Reference Value will return a list of the values of that particular field for all vectors passed to it from the Criteria. If the square brackets and enclosed  $Criteria$  is omitted, the set of vectors will be assumed depending on context. If the Reference value appears in non-vector specific context—in other words, in the main part of a criteria in the CORRECT list—all available vectors will be used. If it is in a vector specific context, such as a message of a Criteria, a Value of a vector field, or the display panel (i.e. everywhere else) it will refer to the currently identified vector.

### 2.2.3.3 Concatenation value

The concatenation value was seen in the example above. It obtains the string values from two or more values and concatenates them. It takes the form:

$string[value1 + value2 \{+ value3 \{+ value4 \{...\}\}\}]$

It is identified by 'string' at the beginning and can contain any combination of any type of Values.

### 2.2.3.4 Math Value

The Math Value works similar to its counterpart in KinimeticPADs. It begins with an optional function, followed by a list of values in square brackets.

$\{function\}[value1 op value2 \{op value3 \{op value4 \{...\}\}\}]$

$function$  is an optional function, such as sine or absolute value operating on the result of the expression in the square brackets. Each  $value$  is any valid value, whether a number, a reference value, or another mathematical expression.  $op$  is an operator. Valid operators are:

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Raise to a power

Example: “[startX\*transY – [startY\*transX]]” would calculate the cross product of the start position with the displacement. If one of the values in an operation is single valued and the other multi-valued (e.g. from a Reference Value operating on multiple vectors) the single-valued value will be multiplied/added/etc. to each value of the other, resulting in a multi-valued result. If both are multi-valued, the first value will be combined with the first, the second with the second, and so forth. It is possible to combine multi-valued items with different numbers of values, but not recommended.

NOTE: the mathematical operations DO NOT RESPECT ORDER OF OPERATION. Instead they are simply evaluated left to right. “[startX\*transY – startY\*transX]” would not give the same result as the example in the previous paragraph, since the startY value would be subtracted first from the product of startX and startY before the result was multiplied by transX. That is why nesting brackets were used.

Valid  $functions$  are listed in the table below.

abs	Take the absolute value.
acos	Compute the inverse cosine (in radians or degrees, depending on UseDegree).
asin	Compute the inverse sine (in radians or degrees, depending on UseDegree).
atan	Compute the inverse tangent (in radians or degrees, depending on UseDegree).
average	Calculate the average of the multiple values calculated inside the expression. Returns a single value.
cos	Compute the cosine (in radians or degrees, depending on UseDegree).
count	Count the number of values calculated inside the expression. Returns a single value.
exp	Calculate the exponential function of the expression ( $e^x$ ).
first	Return the first value in a set of values.
last	Return the last value in a set of values.
log	Calculate the natural logarithm of the expression.
math	No effect (same as nothing present).
max	Return the maximum value from a set of values.
min	Return the minimum value from a set of values.
round	Rounds values to the nearest integer.
sign	Returns 1 if the value is positive, 0 if it is zero, and -1 if negative.
sin	Compute the sine (in radians or degrees, depending on UseDegree).
sqrt	Compute the square root of the expression.
sum	Returns the sum of all the values in the expression.
tan	Compute the tangent (in radians or degrees, depending on UseDegree).

**Table 5: Mathematical functions in VectorPAD**

To get the magnitude of the displacement of a vector, one could specify  $\text{sqrt}[\text{transX}^2 + \text{transY}^2]$  (of course this is a built-in field,  $\text{transR}$ ).

### 2.2.3.5 On Value

The On Value provides a conditional value. Depending on the value of one Value, the On Value returns the value from a list. It takes the form:

$\text{on}[\text{condition}, \text{value0}, \text{value1} \{, \text{value2} \{ \dots \} \}]$

The Value specified by *condition* is evaluated as a number. If it evaluates as zero or negative, the On Value returns the value of *value0*. If it evaluates to one (1), the On Value returns the value of *value1*, and so forth until the end of the list. If *condition* evaluates to a value greater than the index of the last value, the value of the last one in the list is returned.

### 2.2.3.6 Memory Values

The storage values provide a way of storing the result of one calculation for latter use. This is particularly useful in the CORRECT statement, so that subsequent statements can depend on an earlier condition without having to repeat a complex calculation. There are two Memory Values,

$\text{store}[\text{key}, \text{value}]$ ,  $\text{recall}[\text{key}]$

When Store is encountered in an evaluation, it will evaluate the Value represented by *key* for a string, and then store both the current string and numerical values of *value* in a special namespace that persists for the duration of the applet's life. Store then returns the value of *value* unaffected to whatever encapsulates it. At any later point, the value can be recalled by using the Recall Value which specifies the key (name) of the item to recall. Note that while *key* may be any value, it is generally a good idea to use a static string value; otherwise it can easily become very confusing as to what key is being used. One should also be very careful about using memory values as values of fields of a particular vector, since, unlike the CORRECT list, there is no guarantee of order of evaluation.

## 2.2.4 Criteria

Criteria have been referred to above. They have two main functions; to specify conditions that must be satisfied for the applet to evaluate as correct, and to provide encapsulating Values and Criteria with lists of the vectors that satisfy certain conditions. All Criteria take the form

$$\{modifier\} condition \{, correctMessage \{, incorrectMessage\}\}$$

The *modifier* will be discussed below. If condition (with modification) evaluates as true, the *correctMessage* (if specified) will be added to the feedback string, if not the *incorrectMessage* (if specified) will be added. If the Criteria is inside another Value or Criteria, it also provides a list of all the vectors that satisfied the *condition*. There are four types of Criteria: Reference, Range, String and Conditional.

### 2.2.4.1 Reference Criteria

Reference Criteria are the starting type for all evaluations. There are two types, Self and List. Self is the default type if no Criteria is specified in a Reference value. If it is in a vector specific context, such as a message of a Criteria, a Value of a vector field, or the display panel, it will refer to the current vector. If it appears in non-vector specific context—in other words, in the main part of a criteria in the CORRECT list—it has the same result as the List Criteria, which is to return all the available vectors. Both Self and List will always evaluate to true as a condition, so they are mainly used for inside Reference Values.

### 2.2.4.2 Range Criteria

Range Criteria evaluate a condition that a value lies within a specified range. The *range* is specified by one or two values and the comparison operators, '=', '>', '<' and '+/-' to specify equal, greater and less than, and tolerance range. Examples:

value1 = value2 +/- value3	value1 between value2-value3 and value2+value3
value1 > value2 < value3	value1 greater than value2 and less than value3
value1 > value2	value1 greater than value2
value1 < value2	value1 less than value2

Order of greater than and less than signs does not matter. If there are duplicates, the last one is the one that matters. If the equal sign is used, a tolerance MUST be specified, otherwise the tolerance will sometimes be taken to be zero, and any number will fail. The applet will not validity check as to if an impossible range is specified, such as *value* >3 <2. (This can actually be useful at times in combinations with modifiers.)

If any of the *values* specified return multiple values, then multiple comparisons will be made. The default is that if *value1* contains multiple values and *value2* and *value3* contain only one, each value of *value1* will be evaluated. If *value1* contains a single value and *value2* and/or *value3* contain multiple values, *value1* will be compared to each range specified by *value2* and *value3*. If both *value1* and the range specified contain multiple values, the first will be compared to the first, the second to the second, and so forth. If one has fewer values than the other does, it will start over again with the first value, but it is not recommended to specify Criteria where there will be different numbers of multiple values. The default behavior can be modified using the modifiers.

### 2.2.4.3 String criteria

String criteria compare two strings,

$$\text{StringValue1 comparison StringValue2}$$

Each of the four comparisons has a case-sensitive and a case insensitive version; the case sensitive ends with "Case":

Equals / EqualsCase	Two strings match in entirety
BeginsWith / BeginsWithCase	Second string matches first part of first.
EndsWith / EndsWithCase	Second string matches last part of first.
Contains / ContainsCase	Second string appears somewhere in the first.

#### 2.2.4.4 Logical

Logical Criteria combine other Criteria together. They have the form

$$\textit{Criteria1 operation Criteria2}$$

where *Criteria1* and *Criteria2* are any Criteria (though rarely a reference type) and the *operation* is AND, OR or NOT, specified by the symbols “&&”, “||” and “!”. AND will evaluate true if both Criteria are true, and will return all vectors that met both criteria. OR will evaluate if on or the other Criteria are true, and will return all the vectors that met one criteria or the other. NOT will return true if the first condition is true but not the second, and will return all the vectors that met the first criteria but not the second.

#### 2.2.4.5 Modifiers

Optional modifiers go before the condition and may affect the result of evaluation (whether it evaluates true or false), what vectors get passed from the evaluation, and/or how the criteria handles comparisons (range and string) when both the first value and the conditions to meet have multiple values. The possible modifiers, what they affect, and a description are summarized in Table 6.

Modifier	Affects	Description
all	Result	If any values are multiple, all comparisons must be satisfied.
some	Result	If any values are multiple, at least one comparison must be satisfied.
none	Result	No comparison may be satisfied.
not	Result, vectors	Inverts result of comparison, returns vectors not satisfying condition.
before	Vectors	Returns all vectors in the list before the first one that satisfies the condition.
after	Vectors	Returns all vectors in the list after the first one that satisfies the condition
each	Function	Every value must meet every condition.
any	Function	Every value must meet at least one of the conditions.
order	Function	Every value must meet its corresponding condition.

**Table 6: Criteria modifiers in VectorPAD**

### 2.2.5 Criteria and Values together

As one can see, Values can encapsulate other Values and Criteria, and Criteria other Criteria and Values. This makes for flexible but potentially confusing specification. There are three main places that Criteria and Values can be used in VectorPAD. The first is specifying fields of a vector to depend on other conditions. The second is in specifying what to show in the display panel. The third is in specify what constitutes correct in the applet and the provision of context-specific feedback to the user. In this section we will focus on putting them together, mainly through a variety of useful examples.

#### 2.2.5.1 Criteria and Values in RESPONSE

The most common use of Values in RESPONSE was seen earlier in Section 2.2.3; that of specifying the label through a Concatenation Value. Another example which could be used to define a bar to represent the voltage of a point in a circuit is:

```
'V', '1', ",5,10,0,0, size=10 style='verticalBar' color=on[ [sign[transY]+1], 255, 0, 16711680];
```

The first seven positions label the vector as 'V1', establishes its position and that it has an initial value of 0. Size sets the width, and the style specifies a vertical bar. The color uses an On Value; the first math value evaluates to 0 if TransY is negative, 1 if it is zero, and 2 if positive. This then sets the value of 'color' to 255 (blue) if negative, 0 (black) if zero, and 16711680 (red) if positive.

A more complex example follows for a torque diagram. In this example a movable pivot point is provided and the total torque of a system (all the drawn force vectors) around that particular pivot point is calculated and displayed as a vertical bar.

```
'Pivot',", 'pivot',0,0,0,0, mayDelete=false transUnit='m' mayChangeName=false
mayChangeTrans=false mayChangeStart=true size=6 label='Pivot';
```

```
'System', 'Torque',",10,-5,0,0, transUnit='mN' label='System Torque' style=1 mayDelete=false
mayChangeName=false mayChangeTrans=false size=6 color=8355711 TransScale=0.1
TransY=sum[[StartX[transUnit[list] equals 'N']-StartX[Name3[list] equals
'pivot']]*TransY[transUnit[list] equals 'N'] - [[StartY[transUnit[list] equals 'N']-
StartY[Name3[list] equals 'pivot']]*TransX[transUnit[list] equals 'N']];
```

The first vector defined in the response is a black dot labeled 'Pivot' which can be moved around but not otherwise changed. The user can use this to define the position of the pivot about which to calculate the torque. The second vector defined is a gray vertical bar labeled 'System Torque'. It also can be moved but not otherwise changed (directly) by the student. The magic comes in the redefinition of the value for TransY that occupies the second half of the definition. Let's break it down into smaller chunks.

The inner most Criteria of the first part is "transUnit[list] equals 'N'". This gets the transUnit value of all the vectors available (the purpose of 'list' is to override the default of only this vector), and passes on all the ones with the transUnit of "N", which has been set as the default value in the VectorProperties parameter. This has the effect of returning all the vectors (the ones drawn by the student) except the Pivot and System Torque, since they have been defined to have different units.

[[StartX[transUnit[list] equals 'N']] gets the starting horizontal position of all the vectors, excluding the Pivot and System Torque, and then the applet is told to subtract the horizontal position of the Pivot point from each value via StartX[Name3[list] equals 'pivot']. Each item in this list of the horizontal distance from the Pivot to the start of each vector is then multiplied by the vertical displacement of the same vector by \*TransY[transUnit[list] equals 'N'], which uses the same trick to get the vectors drawn by the student. The process is repeated, this time multiplying the vertical displacement of each student-drawn by the horizontal displacement, and taking the difference. This results in a list of values, each of which is the torque produced by a particular vector around the Pivot, which is then summed up.

### 2.2.5.2 Criteria and Values in the Panel

The display of the panel is specified by a series of Values separated by semicolons. The most common values are static strings and reference Values, but all Value types may be used. Reference values will be displayed as editable elements, all other types as display only. A display-only field for a reference value can be obtained by placing the reference inside a STRING value. An editable reference field may be either a text field (default) or a drop down menu. In order to get a drop-down menu, the reference value should be followed by an equal sign and a list of options (strings) inside square brackets. A special value, "\n", specifies a new line in the display. An example of a panel parameter definition and resulting panel is displayed below.

```
<PARAM NAME = "PanelLabels"
Value="Force';name1=['F(G)','F(N)','F(T)','F(f)'];\n;
'of';name2=['Earth','Table','TV','Ground'];\n;
'on';name3=['Earth','Table','TV','Ground'];\n;
string['(+startx+', '+starty+')'];\n;
'Fx:;Transx;\n; 'Fy:;Transy;">
```

The second line is where the definition starts. A static string is supplied as a label and the reference to field name1 is given, specifying "F(G)", "F(N)", "F(T)" and "F(f)" as the options in the drop down menu. The line is terminated by "\n" to indicate to start a new line. Lines 2 and 3 proceed in a similar way, specifying interacting objects. Line 4 displays the starting position of the object, using a String Value to format it with parenthesis and comma, and to make it read-only (but would be automatically updated as the object is moved). The next two lines are labels and reference values, but this time values for a drop-down menu, so editable text boxes are supplied.

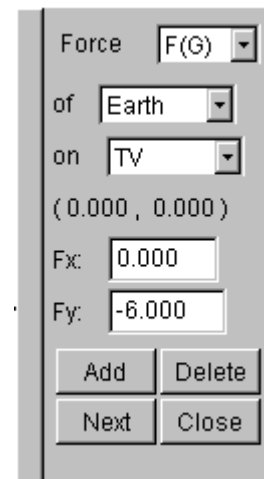


Figure 2: Display panel

### 2.2.5.3 Values and Criteria in Correct

The correct parameter of VectorPAD accepts a list of Criteria. If all Criteria are satisfied, the applet method `isCorrect()` evaluates to TRUE. If any Criteria is not met, it evaluates to FALSE. Any supplied messages will be returned by `computeFeedback()`. Since Values can be nested inside of Criteria and Criteria inside Values, an extremely large range of combinations are possible, beyond what can be discussed in this space. An example will be provided to illustrate how this can be done.

Figure 2 is a simple exercise where student must move the vertical bars to illustrate relative magnitude of voltage at different points and draw the arrows to represent the direction and magnitude of current. The criteria for this to be correct is that  $V1 > V4$ ,  $V3 = V4$ ,  $V2$  the average of  $V1$  and  $V4$ , and the arrows are clockwise and all the same magnitude. The code for some of these criteria is listed below.

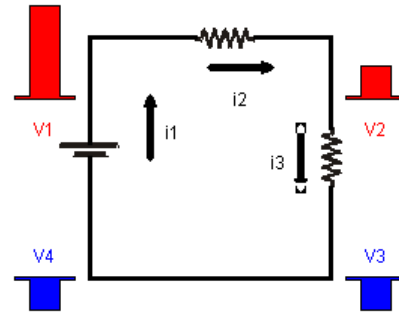


Figure 3: Circuit example

```
transy[name1 equals 'V1'] > transy[name1 equals 'V4'],
    ", 'V1 is at the positive terminal of the battery and V4 at the negative terminal';

transR[name1 beginswith 'i'] = average[transR[name1 beginswith 'i']] +/- 0.1,
    ", 'The same current flows through all elements';

transY[name1 equals 'V2'] = [transY[name1 equals 'V1'] + transY[name1 equals 'V4']/2] +/- 0.1,
    ", 'The potential V2 is not correct';
```

The first Criteria checks that  $V1 > V4$ , providing a message if incorrect. The second checks that the magnitude of the currents is all the same, and the third that  $V2$  is the average of  $V1$  and  $V4$ , again returning an empty message if correct and a string if not. Figure 4 illustrates how the last Criteria is organized. The main criteria on the second line compares several values (specifically that the voltage of  $V2$  is halfway between  $V1$  and  $V4$ ) and returns an empty message if true or 'The potential  $V2$  is not correct' if not. The arrows from the first 'Reference' indicates that it returns the height (`transY`) of objects that satisfy a string criteria, namely that the property 'Name1' equals the value 'V2.' The remaining child values and criteria are similarly illustrated. In a more complex situation, messages could be String Values that might identify more specifically the problem. For example, in the second example more detailed information could be provided in form "Current  $i1$  is less than the average current" or "Current  $i3$  is greater than the average current" by using the following string value as the last message.

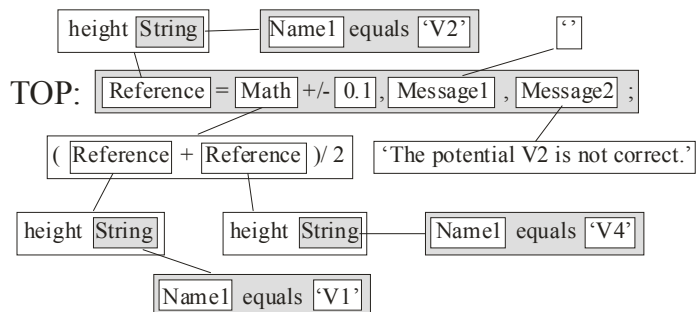


Figure 4: Nestling of Criteria and Values

```
string['Current '+name1+' is '
    +on[[sign[transR -average[transR[name1[list] beginsWith 'i']]]+1],
        'less than', 'equal to', 'more than']
    + ' the average current. ']
```

The first part simply concatenates "Current", the name of the current that failed the check, and "is". The second part is an On Value. The first mathematical value evaluates to 0, 1 or 2 if the current vector is less than, equal to, or greater than the average. Note that this is a vector specific case, so the abbreviate reference value `transR` refers to the particular vector that failed the check and so to get the average of the current vectors we explicitly state "`name1[list]`." The sign function returns -1, 0 or +1 depending on sign of the difference, and one is added to that to get it to the range needed for the on statement, 0-2. The third line finishes up the On Value with the strings to be returned in each case, and the last line finishes the rest

of the sentence. (Note that the On Value will never evaluate to “equal to” if this is the incorrect message, since if the Criteria is satisfied, the correct message will be returned instead of this one.)