

MaSH Environment graphics

Andrew Rock
School of Information and Communication Technology
Griffith University
Nathan, Queensland, 4111, Australia
a.rock@griffith.edu.au

June 9, 2011

Contents

1 Purpose	2
2 Rewrites	2
3 Setting up the window	3
3.1 Purpose	3
3.2 Methods	3
4 Content area dimensions	4
4.1 Purpose	4
4.2 Methods	4
5 Drawing	4
5.1 Purpose	4
5.2 Methods	4
6 Loading images	6
6.1 Methods	6
7 Animation	6
7.1 Methods	6
8 Sounds	7
8.1 Purpose	7
8.2 Methods	7

9 Using the mouse	7
9.1 Purpose	7
9.2 Rewrites	8
9.3 Methods	8
10 Keyboard events	9
10.1 Purpose	9
10.2 Rewrites	9
11 Waiting for fixed times	9
11.1 Methods	9
12 Console input	9
12.1 Purpose	9
12.2 Methods	10
13 Console output	11
13.1 Purpose	11
13.2 Methods	12
14 Math	12
14.1 Purpose	12
14.2 Constants	13
14.3 Methods	13
15 Strings	15
15.1 Purpose	15
15.2 Methods	16
16 Terminating a program	17
16.1 Methods	17
17 Debugger	18
17.1 Purpose	18

1 Purpose

This environment supports drawing in a window, and responding to mouse and keyboard events. It also supports console input and output.

2 Rewrites

```
mandatory
void main ()
```

Purpose: A graphics program must have a `main` method (a procedure with no arguments). This will be the first method to execute. `mashc` automatically rewrites this method to conform to standard Java.

In the `main` method, a program must at least call `setFrameVisible(true)` to make the window visible.

The `main` method, and any method called by `main` may not call any of the drawing methods provided by this environment.

The `main` method may call `animate(int)` just once to start the automatic redrawing of the window for animation.

mandatory

```
void paintWindow ()
```

Purpose: This is the method called to draw the graphics. Every graphics program must define this procedure. The drawing methods provided by this environment may only be called from this method or from a method called from this method.

Do not call this method directly from `main`. It will be called when the window is made visible and then any time the window needs to be redrawn.

3 Setting up the window

3.1 Purpose

Use these methods to set up the window into which all drawing is done.

In Java, the window is known as a `Frame`.

3.2 Methods

```
void setFrameSize (int width, int height)
```

Purpose: Sets the size of the window with `width` and `height`, both in pixels. Call this before making the window visible.

```
void setFrameTitle (String title)
```

Purpose: Set the `title` of the window. Call this before making the window visible.

```
void setFrameVisible (boolean visibility)
```

Purpose: Set the `visibility` of the window. `true` means visible.

4 Content area dimensions

4.1 Purpose

The size of the window, set with `setFrameSize(int, int)`, is the size of the whole window including its title bar and resizing control areas. The actual size of the content area into which we can draw is platform dependent, and the as the user resizes the window may change. These methods allow the program to find out the true current dimensions of the content area.

4.2 Methods

```
int contentWidth ()
```

Purpose: This returns the current width of the content area of the frame in pixels.

```
int contentHeight ()
```

Purpose: This returns the current height of the content area of the frame in pixels.

5 Drawing

5.1 Purpose

Use these methods draw in the window.

These methods may only be called by the `paintWindow()` method or by any method called by `paintWindow()`.

All lengths are in pixels. All angles are in degrees.

All rectangles are specified by `(x, y, width, height)`, where `(x,y)` are the coordinates of the top, left corner of the rectangle.

5.2 Methods

```
void clearRect (int x, int y, int width, int height)
```

Purpose: Clear the rectangle by filling it with the background surface.

```
void drawArc (int x, int y, int width, int height, int startAngle,  
int arcAngle)
```

Purpose: Draws the outline of an elliptical arc inscribed in the specified rectangle, starting at `startAngle` degrees (0 is 3 o'clock), for `arcAngle` degrees (positive is anticlockwise).

```
void drawLine (int x1, int y1, int x2, int y2)
```

Purpose: Draw a line from (x_1, y_1) to (x_2, y_2) .

```
void drawOval (int x, int y, int width, int height)
```

Purpose: Draws the outline of an oval inscribed in the specified rectangle.

```
void drawPolygon (int[] xpoints, int[] ypoints, int npoints)
```

Purpose: Draws the outline of a closed polygon with `npoints` vertices with coordinates $(xpoints[i], ypoints[i])$.

```
void drawPolyline (int[] xpoints, int[] ypoints, int npoints)
```

Purpose: Draws a sequence of `npoints - 1` connected lines with `npoints` end points at coordinates $(xpoints[i], ypoints[i])$.

```
void drawRect (int x, int y, int width, int height)
```

Purpose: Draws the outline of the specified rectangle.

```
void drawRoundRect (int x, int y, int width, int height, int arcWidth,  
int arcHeight)
```

Purpose: Draws the outline of the specified rectangle with rounded corners made with arcs with width `arcWidth` and height `arcHeight`.

```
void drawString (String s, int x, int y)
```

Purpose: Draw string `s` with (x, y) being the coordinates of the bottom-left corner of the first character.

```
void fillArc (int x, int y, int width, int height, int startAngle,  
int arcAngle)
```

Purpose: Fills an elliptical arc inscribed in the specified rectangle, starting at `startAngle` degrees (0 is 3 o'clock), for `arcAngle` degrees (positive is anticlockwise).

```
void fillOval (int x, int y, int width, int height)
```

Purpose: Fills an oval inscribed in the specified rectangle.

```
void fillPolygon (int[] xpoints, int[] ypoints, int npoints)
```

Purpose: Fills a closed polygon with `npoints` vertices with coordinates $(xpoints[i], ypoints[i])$.

```
void fillRect (int x, int y, int width, int height)
```

Purpose: Fills the specified rectangle.

```
void fillRoundRect (int x, int y, int width, int height, int arcWidth,  
int arcHeight)
```

Purpose: Fills the specified rectangle with rounded corners made with arcs with width `arcWidth` and height `arcHeight`.

```
void setColor (int red, int green, int blue)
```

Purpose: Set the color to draw with by defining the amounts of (`red`, `green`, `blue`). (0, 0, 0) is black. (255, 255, 255) is white.

Precondition: $0 \leq \text{red} \leq 255$; $0 \leq \text{green} \leq 255$; and $0 \leq \text{blue} \leq 255$.

```
void drawImage (int imageID, int x, int y)
```

Purpose: Draw the image selected by `imageID` with (x, y) being the coordinates of its top-left corner.

Precondition: `imageID` must refer to an image that has been preloaded with `getImage(String)`.

6 Loading images

6.1 Methods

```
int getImage (String path)
```

Purpose: Load an image from the file at `path` and return its new ID. If the ID returned is negative, the load failed.

7 Animation

7.1 Methods

```
void repaint ()
```

Purpose: Causes the window to be repainted.

```
void animate (int fps)
```

Purpose: Causes the window to be repainted at `fps` frames per second. Only call this method *once*.

Precondition: `fps > 0`.

8 Sounds

8.1 Purpose

Like images, sound files can be loaded and played many times. File formats include `.wav` and `.aif`. Others may work. Loaded sounds are identified by an ID number.

A sound once loaded, can be played or looped. If it is played again, before it is finished, it will start from the beginning again. Multiple sounds may be played at the same time. A single sound file loaded multiple times may be played at the same time.

8.2 Methods

```
int getSound (String url)
```

Purpose: Load a sound clip from the file at `url` and return its new ID. If the ID returned is negative, the load failed.

Precondition: `url` is a URL, not just a file name. For example, `file:sound.wav`.

```
void playSound (int soundID)
```

Purpose: Play the sound selected by `soundID`.

Precondition: `soundID` must refer to a sound that has been preloaded with `getSound(String)`.

```
void loopSound (int soundID)
```

Purpose: Loop the sound selected by `soundID`.

Precondition: `soundID` must refer to a sound that has been preloaded with `getSound(String)`.

```
void stopSound (int soundID)
```

Purpose: Stop the sound selected by `soundID`.

Precondition: `soundID` must refer to a sound that has been preloaded with `getSound(String)`.

9 Using the mouse

9.1 Purpose

Implement the rewrite methods to handle mouse events that occur within the content area of the frame. The functions provide the mouse position at any time.

9.2 Rewrites

`void onMouseClicked (int x, int y)`

Purpose: Implement this method to handle mouse clicks in the content area of the frame.

(x, y) will be the passed the coordinates of the mouse relative to the content area of the frame at the time the event occurred.

`void onMouseEntered (int x, int y)`

Purpose: Implement this method to handle the mouse entering the content area of the frame.

(x, y) will be the passed the coordinates of the mouse relative to the content area of the frame at the time the event occurred.

`void onMouseExited (int x, int y)`

Purpose: Implement this method to handle the mouse exiting the content area of the frame.

(x, y) will be the passed the coordinates of the mouse relative to the content area of the frame at the time the event occurred.

`void onMousePressed (int x, int y)`

Purpose: Implement this method to handle the mouse being pressed within the content area of the frame.

(x, y) will be the passed the coordinates of the mouse relative to the content area of the frame at the time the event occurred.

`void onMouseReleased (int x, int y)`

Purpose: Implement this method to handle the mouse being released within the content area of the frame.

(x, y) will be the passed the coordinates of the mouse relative to the content area of the frame at the time the event occurred.

9.3 Methods

`int mouseX ()`

Purpose: Returns the current horizontal ordinate x of the mouse pointer relative to the content area of the frame.

`int mouseY ()`

Purpose: Returns the current vertical ordinate y of the mouse pointer relative to the content area of the frame.

10 Keyboard events

10.1 Purpose

Implement the rewrite methods to handle keyboard events when the frame is in focus.

The codes that identify keys may be discovered by experiment. (Write handlers that print the codes to the console.) Some handy codes are those for the arrow keys: left = 37; up = 38; right = 39; and down = 40.

10.2 Rewrites

```
void onKeyPressed (int code)
```

Purpose: Implement this method to handle a key being pressed while the frame is in focus.

code will be the passed a code that identifies the key.

```
void onKeyReleased (int code)
```

Purpose: Implement this method to handle a key being released while the frame is in focus.

code will be the passed a code that identifies the key.

```
void onKeyTyped (char c)
```

Purpose: Implement this method to handle a key being typed while the frame is in focus.

c will be the passed the character that was typed.

11 Waiting for fixed times

11.1 Methods

```
void sleep (int ms)
```

Purpose: Makes the program wait for a requested number of ms (milliseconds).

12 Console input

12.1 Purpose

The following are methods for reading from standard input (which is usually the keyboard) and for checking to see if there is more input to read.

12.2 Methods

`int readInt ()`

Purpose: Returns the next integer from standard input.

Precondition: Will cause a run time error unless there is an integer in standard input to read. Use `isNextInt()` to check first.

`long readLong ()`

Purpose: Returns the next long from standard input.

Precondition: Will cause a run time error unless there is a long in standard input to read. Use `isNextLong()` to check first.

`boolean readBoolean ()`

Purpose: Returns the next boolean from standard input.

Precondition: Will cause a run time error unless there is a boolean in standard input to read. Use `isNextBoolean()` to check first.

`double readDouble ()`

Purpose: Returns the next double from standard input.

Precondition: Will cause a run time error unless there is a double in standard input to read. Use `isNextDouble()` to check first.

`float readFloat ()`

Purpose: Returns the next float from standard input.

Precondition: Will cause a run time error unless there is a float in standard input to read. Use `isNextFloat()` to check first.

`String readWord ()`

Purpose: Returns the next word as a `String`. A “word” is a sequence of one-or-more non-whitespace characters.

Precondition: Will cause a run time error unless there is a word in standard input to read. Use `isNextWord()` to check first.

`String readLine ()`

Purpose: Returns the next line of text as a `String`. A line is a sequence of zero-or-more characters terminated by the end of line, which is not returned as part of the line.

Precondition: Will cause a run time error unless there is a line in standard input to read. Use `isNextLine()` to check first.

`boolean isNextInt ()`

Purpose: Returns `true` if and only if there is an integer in standard input available to read, that is `readInt()` would succeed.

`boolean isNextLong ()`

Purpose: Returns `true` if and only if there is a long in standard input available to read, that is `readLong()` would succeed.

`boolean isNextBoolean ()`

Purpose: Returns `true` if and only if there is a boolean in standard input available to read, that is `readBoolean()` would succeed.

`boolean isNextDouble ()`

Purpose: Returns `true` if and only if there is a double in standard input available to read, that is `readDouble()` would succeed.

`boolean isNextFloat ()`

Purpose: Returns `true` if and only if there is a float in standard input available to read, that is `readFloat()` would succeed.

`boolean isNextWord ()`

Purpose: Returns `true` if and only if there is a word in standard input available to read, that is `readWord()` would succeed.

`boolean isNextLine ()`

Purpose: Returns `true` if and only if there is a line in standard input available to read, that is `readLine()` would succeed.

13 Console output

13.1 Purpose

The following are methods for printing to standard output (which is usually the terminal window).

13.2 Methods

`void print (char c)`

Purpose: Writes character `c` to standard output.

`void print (String s)`

Purpose: Writes string `s` to standard output.

`void print (long i)`

Purpose: Writes integral `i` to standard output.

`void print (boolean p)`

Purpose: Writes boolean `p` to standard output.

`void print (double x)`

Purpose: Writes floating point number `x` to standard output.

`void println (char c)`

Purpose: Writes character `c` and then a newline to standard output.

`void println ()`

Purpose: Writes a newline to standard output.

`void println (String s)`

Purpose: Writes string `s` and then a newline to standard output.

`void println (long i)`

Purpose: Writes integral `i` and then a newline to standard output.

`void println (boolean p)`

Purpose: Writes boolean `p` and then a newline to standard output.

`void println (double x)`

Purpose: Writes floating point number `x` and then a newline to standard output.

14 Math

14.1 Purpose

The following are some commonly used numeric constants and functions.

14.2 Constants

`final int MAX_INT`

Purpose: A constant holding the maximum value an `int` can have, $2^{31} - 1$.

`final int MIN_INT`

Purpose: A constant holding the minimum value an `int` can have, -2^{31} .

`final long MAX_LONG`

Purpose: A constant holding the maximum value a `long` can have, $2^{63} - 1$.

`final long MIN_LONG`

Purpose: A constant holding the minimum value a `long` can have, -2^{63} .

`final double PI`

Purpose: The closest `double` approximation to π .

14.3 Methods

`double abs (double a)`

Purpose: Returns the absolute value of `a`.

`float abs (float a)`

Purpose: Returns the absolute value of `a`.

`long abs (long a)`

Purpose: Returns the absolute value of `a`.

`int abs (int a)`

Purpose: Returns the absolute value of `a`.

`double ceil (double a)`

Purpose: Returns the least double value that is greater than or equal to `a` and equal to an integer.

`double exp (double x)`

Purpose: Returns e^x , that is Euler's constant e raised to power x .

`double floor (double a)`

Purpose: Returns the greatest double value that is less than or equal to `a` and equal to an integer.

`double log (double x)`

Purpose: Returns the natural logarithm of x .

`long round (double a)`

Purpose: Returns the closest `long` to `a`.

`int round (float a)`

Purpose: Returns the closest `int` to `a`.

`double sqrt (double a)`

Purpose: Returns the square root of `a`.

Precondition: $a \geq 0.0$.

`double pow (double a, double b)`

Purpose: Returns `a` raised to the power `b`, a^b .

`double sin (double a)`

Purpose: Returns the trigonometric sine of `a` radians.

`double cos (double a)`

Purpose: Returns the trigonometric cosine of `a` radians.

`double tan (double a)`

Purpose: Returns the trigonometric tangent of `a` radians.

`double asin (double a)`

Purpose: Returns the trigonometric arc sine of `a` in radians.

`double acos (double a)`

Purpose: Returns the trigonometric arc cosine of `a` in radians.

`double atan (double a)`

Purpose: Returns the trigonometric arc tangent of **a** in radians.

`double max (double a, double b)`

Purpose: Returns the greater of **a** and **b**.

`float max (float a, float b)`

Purpose: Returns the greater of **a** and **b**.

`int max (int a, int b)`

Purpose: Returns the greater of **a** and **b**.

`long max (long a, long b)`

Purpose: Returns the greater of **a** and **b**.

`double min (double a, double b)`

Purpose: Returns the lesser of **a** and **b**.

`float min (float a, float b)`

Purpose: Returns the lesser of **a** and **b**.

`int min (int a, int b)`

Purpose: Returns the lesser of **a** and **b**.

`long min (long a, long b)`

Purpose: Returns the lesser of **a** and **b**.

`double random ()`

Purpose: Returns a random value x such that $0.0 \leq x < 1.0$.

15 Strings

15.1 Purpose

The following are methods for working with **Strings**.

15.2 Methods

`int length (String s)`

Purpose: Returns the length of `s`.

`char charAt (String s, int i)`

Purpose: Returns the character at position `i` in `s`.

Precondition: $0 \leq i < \text{length}(s)$.

`boolean equals (String a, String b)`

Purpose: Returns `true` if and only if `a` contains the same sequence of characters as in `b`.

`String format (boolean p, int w)`

Purpose: Returns `p` converted to a string, padded with spaces to a minimum width $|w|$. If `w` is negative, the result is left-justified, otherwise right-justified.

Examples:

`format(true, 10)` returns " true"; and

`format(true, -10)` returns "true".

`String format (char c, int w)`

Purpose: Returns `c` converted to a string, padded with spaces to a minimum width $|w|$. If `w` is negative, the result is left-justified, otherwise right-justified.

Examples:

`format('a', 3)` returns " a"; and

`format('a', -3)` returns "a".

`String format (double d, int w, char f, int p)`

Purpose: Returns `d` converted to a string, padded with spaces to a minimum width $|w|$. If `w` is negative, the result is left-justified, otherwise right-justified. `f` controls the format: `'e'` selects scientific notation; `'f'` selects fixed point; or `'g'` selects the best format depending on the number and `p`. For `'e'` and `'f'`, `p` is the number of decimal digits to display after the decimal point, but for `'g'` it is the total number of digits.

Examples:

`format(1234.56789, 12, 'e', 4)` returns " 1.2346e+03";

`format(1234.56789, 12, 'f', 4)` returns " 1234.5679";

```
format(1234.56789, 12, 'g', 4) returns "      1235";
format(0.000001234567, 12, 'e', 4) returns " 1.2346e-06";
format(0.000001234567, 12, 'f', 4) returns "      0.0000";
and
format(0.000001234567, 12, 'g', 4) returns " 1.235e-06".
```

`String format (long l, int w)`

Purpose: Returns `l` converted to a string, padded with spaces to a minimum width `|w|`. If `w` is negative, the result is left-justified, otherwise right-justified.

Examples:

```
format(42, 5) returns "   42"; and
format(42, -5) returns "42   ".
```

`String format (String s, int w)`

Purpose: Returns `s` padded with spaces to a minimum width `|w|`. If `w` is negative, the result is left-justified, otherwise right-justified.

Examples:

```
format("aaa", 5) returns "  aaa"; and
format("aaa", -5) returns "aaa  ".
```

`boolean parseBoolean (String s)`

Purpose: Returns `s` converted to a boolean.

`int parseInt (String s)`

Purpose: Returns `s` converted to an int.

`long parseLong (String s)`

Purpose: Returns `s` converted to a long.

`float parseFloat (String s)`

Purpose: Returns `s` converted to a float.

`double parseDouble (String s)`

Purpose: Returns `s` converted to a double.

16 Terminating a program

16.1 Methods

`void exit ()`

Purpose: Terminates the program.

17 Debugger

17.1 Purpose

This environment supports a debugger with a graphical user interface. It allows the user to slow down the execution of a program, so that the flow of control through the program may be observed and the values in variables monitored.

To activate the debugger, compile your program with the `mashc +debug` option.

While the program is running under the control of the debugger, the original MaSH program's source code is displayed on the left, with a pointer showing the statement that is about to execute. Sometimes it will point to a closing brace, indicating that the current method is about to be exited, or a loop guard is about to be tested again, depending on the context.

The contents of all the program's variables are displayed on the right in a table. The entries in the *scope* column, consist of a number and a word. The number indicates which method invocation the variable belongs to. For no-method MaSH programs the number is always 0. For method MaSH programs, 0 indicates a global variable, 1 indicates the `main` method, 2 indicates a method called from `main`, 3 indicates a method called from 2, etc. The word is blank for variables in a no-method program, unless the variable is declared in a block, which makes it *local*. In method programs, the word can be: *global*; *local*; or *param*. Variables are added to the table as they come into scope and removed again when their enclosing block exits. Anything changed on the last program step is displayed in green.

Limitations: The debugger can not display values of arrays with 3 or more dimensions. The debugger can not step through methods that run in the programs event dispatch thread (for example, the `paintWindow` method in the `graphics` environment). This GUI debugger can not work in environments without a big screen (for example, Lego Mindstorms).

Index

abs, 13
acos, 14
animate, 6
asin, 14
atan, 14

ceil, 13
charAt, 16
clearRect, 4
contentHeight, 4
contentWidth, 4
cos, 14

drawArc, 4
drawImage, 6
drawLine, 5
drawOval, 5
drawPolygon, 5
drawPolyline, 5
drawRect, 5
drawRoundRect, 5
drawString, 5

equals, 16
exit, 17
exp, 13

fillArc, 5
fillOval, 5
fillPolygon, 5
fillRect, 6
fillRoundRect, 6
floor, 14
format, 16, 17

getImage, 6
getSound, 7

isNextBoolean, 11
isNextDouble, 11
isNextFloat, 11
isNextInt, 11
isNextLine, 11
isNextLong, 11
isNextWord, 11

length, 16
log, 14
loopSound, 7

main, 2
max, 15
MAX_INT, 13
MAX_LONG, 13
min, 15
MIN_INT, 13
MIN_LONG, 13
mouseX, 8
mouseY, 8

onKeyPressed, 9
onKeyReleased, 9
onKeyTyped, 9
onMouseClicked, 8
onMouseEntered, 8
onMouseExited, 8
onMousePressed, 8
onMouseReleased, 8

paintWindow, 3
parseBoolean, 17
parseDouble, 17
parseFloat, 17
parseInt, 17
parseLong, 17
PI, 13
playSound, 7
pow, 14
print, 12
println, 12

random, 15
readBoolean, 10
readDouble, 10
readFloat, 10
readInt, 10

readLine, 10
readLong, 10
readWord, 10
repaint, 6
round, 14

setColor, 6
setFrameSize, 3
setFrameTitle, 3
setFrameVisible, 3
sin, 14
sleep, 9
sqrt, 14
stopSound, 7

tan, 14