Magic Number Machine Key Features

Machine Number Machine is a graphically-displayed, high-precision, scientific calculator featuring:

- **25 digits of precision.**
  Using the "Disp" button you can increase the precision up to 25 digits or apply scientific notation or fixed precision.

- **Full expression view, graphically laid out.**
  So that 2 over 5 appears as a 2, over a 5. You can also change the insertion point to go back and correct mistakes.

- **Hexadecimal, binary, decimal and octal display including fractional, scientific notation and negative numbers in each.**
  Great.

- **Complex numbers.**
  For people who know how to pronounce Euler. [Hint: doesn't rhyme with Bueller].

- **Data values (memory to store values) and statistical functions to apply to them.**
  Hit the "Data" button to add something to data. Then use the "Data" drawer to apply functions to the data.

- **Array functions.**
  Because humans are bad at solving simultaneous equations.

- **An expression history.**
  Go back to anything.

- **Large number of scientific constants available.**
  If you want other constants, open up the source code and change them.

- ~~**Totally bug free.**~~ **Plenty of bugs.**
  You'll never run out.

If the back button on your browser is too far away, you can click here to go back to the Help Contents.

Magic Number Machine Buttons

The following image shows all the keyboard shortcuts for the buttons in the user interface:

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
shortcuts: same as the digit

These buttons cause digits to be entered at the current input point. Depending on the radix (base of the number system) currently selected, any of the digits from 2 to F may be disabled—if they are available, they will become active.

Operators: +, -, ×, ÷
shortcuts: +, -, *, / respectively

These buttons insert their mathematical operators into the expression at the current input point. The buttons group numbers according to order of operations as though the entire expression had been written on one line.
i.e., 2 + 5 / 6 * 7 gives 2 + (5 / (6 * 7))

Modifiers: (+/-), ·, EE
shortcuts: Command-'-', ., x respectively (that's Command-minus, period, letter 'x')

These buttons adjust the current value by changing the sign, appending a fractional point (its only a decimal point if you're in decimal) or appending a "scientific notation"-style exponent.

Clearing and deleting: AC, C, DEL
shortcuts: command-delete, escape and delete respectively

AC clears the entire current expression. C sets the number at the current insertion point to zero. DEL deletes a single digit from the current number or the operator at the insertion point if there is no number.

Constant Numbers: i, pi
shortcuts: i, p respectively

These buttons insert their mathematical value at the current insertion point. If a number is already current, they insert an implied multiply first. This allows you to type 2p and get (2 multiplied by pi).

Brackets: (,)
shortcuts: (,) respectively

These buttons insert brackets at the current insertion point. If the left bracket is pressed and a number is current, an implied multiplication is inserted. This allows you to type "2(3+5)" and get (2 multiplied by (3 + 5)). The right bracket has no effect if the left bracket has not been pressed.

Exponents: e^x, x^y, x^2, x^3, square root, cube root, nth root
shortcuts: z, y, w, v, respectively

e^x causes the next number to be used as an exponent to which the number 'e' (base of the natural logarithm 2.718...) will be raised. If the shift key is pressed first, this button causes the number 10 to be raised instead of e. x^y causes the current number to be raised to the exponent given by the next number. x^2 causes the current number to be squared (raised to the power two). Square root causes the current number to be raised to the power ½.

Logarithms: ln
shortcuts: 1 [lowercase L]

The next number will have its natural logarithm taken. If the shift key is pressed, then the next number will have its base-10 logarithm taken instead.

Trigonometric functions: sin, cos, tan
shortcuts: s, o, t

These buttons cause their respective trigonometric functions to be applied to the next numbers entered. If the shift key is pressed, the inverse functions are applied. If the option key is pressed, the hyperbolic functions are applied. If both the shift and the option keys are pressed, the inverse hyperbolic functions are applied.

Modulo, Rounding: %, Rnd
shortcuts: m

The % button inserts a modulo operator after the current number. This is *not* a "Percentage" button. Modulo causes the number before it to be divided by the number after it and the result is the remainder of this operation. This function also works on fractional number. If the shift key is pressed, this button simply causes the current number to be rounded towards zero (truncates the fractional part).

Permutations and Combinations: nPr, nCr
shortcuts: n

Calculates how many permutations of the preceding number can be taken from the proceeding number. If shift is pressed, calculates how many combinations of the previous number can be taken from the proceeding number. The difference is that a permutation considers the same numbers in a different order to be different, whereas a combination considers only the presence (order does not matter).

Series: x!
shortcuts: q

Calculates the factorial of the current number (all the numbers from 1 up to the number multiplied together). This is only valid for whole numbers greater or equal to zero. If shift is pressed, a sum of all the numbers up to the current number is performed instead.

Exponent shifting: <E
shortcuts: g

Shifts the result three digits to the left and subtracts 3 from the "scientific notation" exponent. If shift is pressed, shifts the result right and adds 3 instead. If the result is not showing, this button will generate the result of the expression.

Logic Operations: and, or
shortcuts: h, j

Performs the binary operations "and" and "or" between the preceding and proceeding numbers. The operation is always performed in binary, even if the current radix is not binary. If you don't know what that means, then maybe these buttons are not for you. If shift is pressed, then binary "not" and binary "xor" respectively are applied to the proceeding numbers.

Complex number functions: Re, abs

shortcuts: k, r

Re gives the real component of the proceeding number. If shift is pressed, the imaginary component is returned instead. abs gives the magnitude of the proceeding complex number (the magnitude is the square root of the real part squared plus the imaginary part squared). If the shift key is pressed, the argument (angle about the origin between the vector of the number plotted on a real versus imaginary cartesian graph and the real positive axis) is returned instead. Wow, that's a clumsy sentence.

Shift buttons: Shft
shortcuts: shift

The shift button toggles "Shift" mode. The shift key on the keyboard enables "Shift" mode while it is pressed. While "Shift" mode is enabled red functions are used where available instead of the buttons they are over. Pressing another button will disable "Shift" mode if it was toggled by pressing the button (if it is enabled by holding down the keyboard shift key then it will remain active until the shift key is released).

Other bits and pieces: Deg, Disp
shortcuts: Command-T, Command-D

Deg cycles the trigonometric mode through degree, radians and gradians.

The Disp button allows you to chose a different display precision (number of digits shown). If the shift key is pressed, the precision chosen is for scientific notation (a specified number of digits of precision). If the option key is pressed, the precision chosen is for fixed notation (a specified number of decimal places).

Data: Add to Data
shortcuts: space key

Append the result of the current expression to the data in the Data drawer.

If the back button on your browser is too far away, you can click here to go back to the Help Contents.

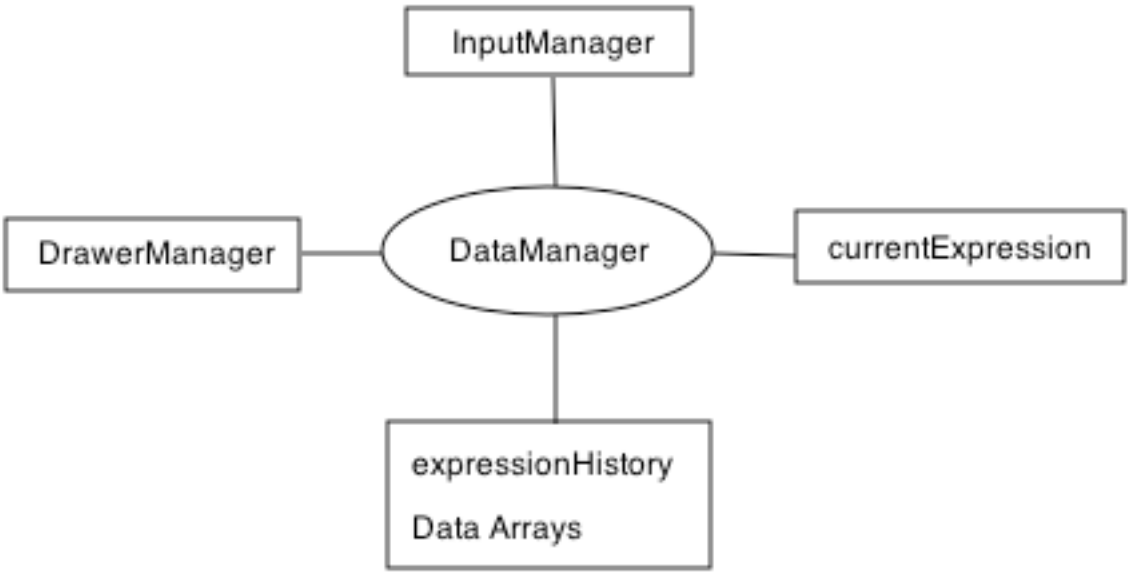
Magic Number Machine Source Code

This document will walk you through the design of the program and how the various components work. If you're just interested in changing a piece of the code, try jumping ahead to the Useful bits to hack.

Alternately, all of the source files have a block comment at the top which explains (tersely) what the file does. You can just dive straight into the code.

Still, if that ain't your thing...

Top Level Design
Internally, the program is structured something like this:



There is always a single DataManager object, a single InputManager and a single DrawerManager. These classes are all created when the program awakes from its NIB file.

DataManager
The DataManager is the hub of the program. It manages most of the persistent data and manages communication between the other two managers, the ExpressionView (the main view in the window) and the currentExpression (which is the data displayed in the ExpressionView).

The DataManager owns most of the important data in the program, including the currentExpression, the expression history and all of the data arrays.

InputManager
All user input through the buttons in the window, and most other methods, gets routed through the InputManager. The InputManager handles the fairly boring tasks of converting button presses into messages sent to the current expression. Ideally, if I was following the model I drew up above, the InputManager would communicate these messages to the DataManager which would then pass them on to the currentExpression but I hate writing pass through interfaces so the InputManager sends the messages to the currentExpression directly.

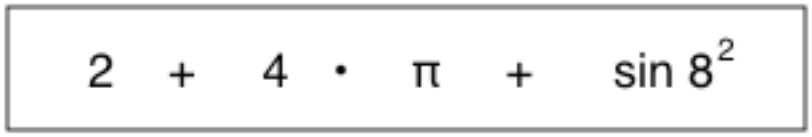
Interesting point... the InputManager inherits from NSView because I shove it into the responder hierarchy as first responder so that it can handle copy and paste from the menu. Why I didn't just use the ExpressionView as the first responder (it would be more correct from a conceptual point of view) is historical more than anything else (the ExpressionView didn't exist when I first put in copy and paste and I was too lazy to do it right).

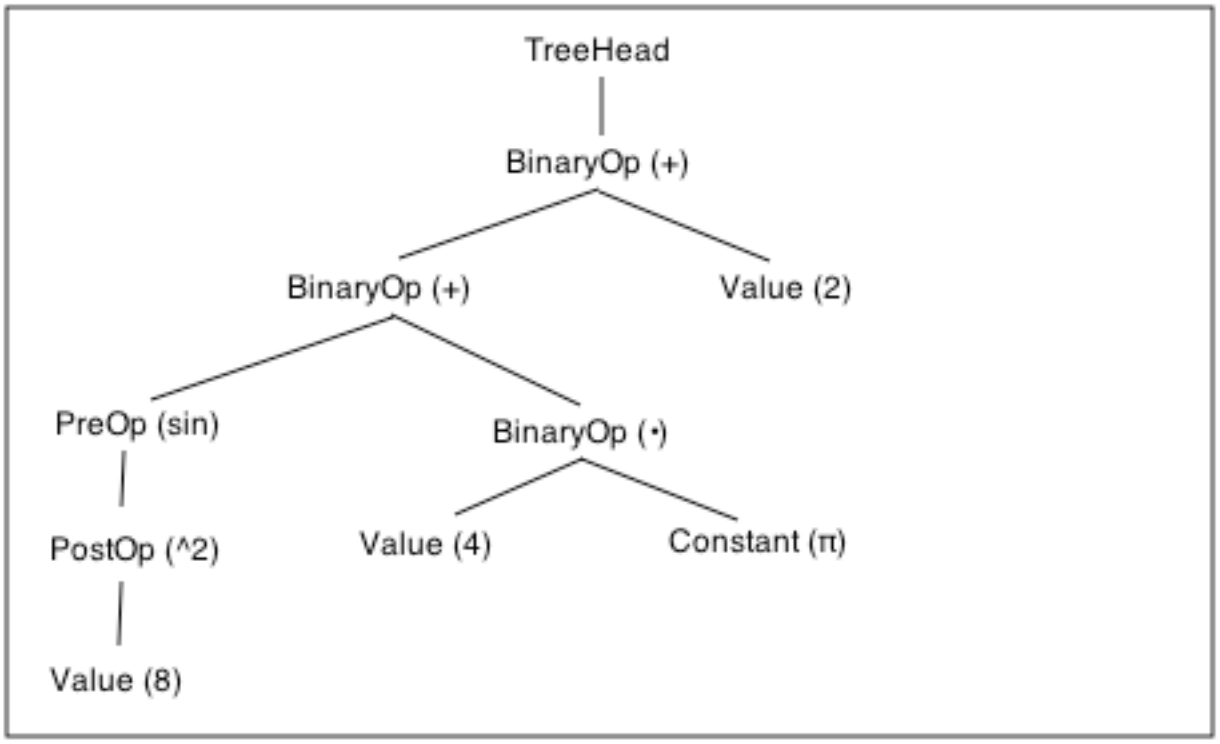
DrawerManager
This is a misguided bucket of a class. It manages both the behaviour and the data associated with the drawers. It moves the drawers in and out (fun) when required, populates the various tables with data and handles user interaction with the expression history and data arrays.

Expressions
In case you haven't worked it out by now, everything displayed in the big white part of the window is called an expression. The view is an ExpressionView but you can work out how it operates yourself; here we will look at how the Expression class works.

Expressions are made by a tree of classes, each of which inherits from Expression. Each node in the tree has one parent (except the TreeHead) and most can have one child. Exceptions to the one child rule are BinaryOp, which is the only type of node that can have two children, and Constant (and its descendent Value) which have no children.

A typical Expression tree will look like this:





The top box shows an expression and the bottom box shows the tree that contains it.

The most difficult part of an expression tree is maintaining the order of operations while allowing the expression to be edited. I've done two things to try to help with this: firstly, as the expression moves left to right across the line, each term gets deeper down the tree (this makes deletion right to left easier).

The second point is that + and - BinaryOps always get appended to the left, whereas •. x. / and % get appended to the right. When things get deleted in the middle, it gets kinda wrong. Have a look at replaceChild in BinaryOp for more.

Interesting to note that there is little internal different between a PreOp and a PostOp – they only really differ by how they are entered.

Useful Bits to Hack

Adding another function to the Data Functions
Insert another object in the dataFunctionRows (roughly line 56 in DrawerManager.m). The object should be defined like this:

```
[NSArray arrayWithObjects:@"Name",
   [NSValue value:&@selector(function:) withObjCType:@encode(SEL)], nil],
```

where "Name" is the name that you want the function as in the data function table. "function" is the name of a function that you should add to DataFunctions.m. The "sum" function is a good one to look at... it is very simple and shows how to perform basic arithmetic and return a result. You can also change data in the array directly if that makes sense.

The 2D and Array functions work in exactly the same way... just add an object to the arrayDataFunctionRows or data2DFunctionRows as appropriate. Interpreting the NSMutableArray "values" is a little harder to work with in these cases. I recommend looking at one of the other functions first to see how its done.

Adding/Changing the Constants in the Constants Drawer
Even easier than above, just add or change a line that looks like this:

```
[NSArray arrayWithObjects:[NSString stringWithUTF8String:"name"],
   [BigCFloat bigFloatWithDouble:123456 radix:10], nil],
```

in the constantsDataRows definition at line 100 of DrawerManager.m. "name" is what will appear in the constants table and 123456 is the value that will be inserted.

Brushed Metal UI
Though I'm sure I don't know why you would...

Open the Magic Number Machine project file. Open the Interface.nib file in the Resources folder. Click on "Main Window" in the Instances window/tab. Select "Show Info" from the Tools menu. Click on the "Textured Window" checkbox.

Save the file, close it and choose "Build and Run" from your development environment.

Implementing division
When implementing BigFloat.m, the single hardest part was implementing the division. Something I didn't realise until I had to do it for myself, is that if you store your number in blocks of bits (BigFloat uses blocks of 16-bit values), you have to set a maximum limit on each block that is a power of your radix minus 1. ie For base 10 numbers in 16 bit, it is important to limit each 16-bit block to 9999. Why does this matter? Without it, you can't do long division.

Division has stacks of other quirks. If you need to implement it yourself and are tired of people telling you to look it up in Knuth, read divideBy in BigFloat.m – it isn't fast but it's straightforward and reasonably self contained.

Getting the Bezier Path of text

Have you ever wondered how? It's hard to work out from documentation alone. Try looking at line 707-718, 753-761 and line 885, all in generateValuePath in Value.m. As of this writing, be careful to explicitly release the NSTextStorage (or else a weird bug will crash your program) and don't use getGlyphs:range: instead of the repeated calls to glyphAtIndex, since it also crashes periodically. I don't think these functions get called much.

Fun (?) homework projects for home viewers like you

Things I haven't implemented but would be really useful for the greater good of the world:

- Fix any bugs you find and submit the fixes back to me
- Have symbols appear instead of numbers for the Constants
- Make the Expression History and Data Arrays persistent so that if you quit the program, they're there when you start next time.

Extra credit (because its really hard)

- Implement a Gamma function for BigCFloat

If the back button on your browser is too far away, you can click here to go back to the Help Contents.

Magic Number Machine RAQ

If I type 5318008 on my pocket calculator and turn it upside down it spells BOOBIES. If I use Magic Number Machine and turn my monitor upside down, I hear a popping sound and the screen goes black. Can you help?

Find a computer where the monitor isn't permanently ruined and try 184594917 in decimal before putting the calculator in hexadecimal mode.

What with all the "reality" television nowadays?
I know! I don't know what the networks consider to be "real" but most of those people are *way* too attractive.

Of course, networks like "reality" TV because it's cheap and the same 10% of people who watch one "reality" show will watch them all -- resulting in reliable ratings. Everyone else gets their cheap porn on the 'net instead and dreams wistfully of a time when a "script" was something given to the "actors", not followed by the publicists.

Actually, I have an idea for a "reality" show... it's about a programmer who pitches an idea for a "reality" TV show to a group of network executives. They give him $1 million prize-money for no reason at all and he gives them the finger. We could draw the whole thing out to 13 episodes with long ad-breaks and fire-side chats that repeat things that just happened.

What is that "·" symbol?
That's an implicit multiplication. You've entered two numbers into the expression and haven't used anything to separate them. The program has assumed you want to multiply them. The symbol is used to remind you what is happening.

i.e. 5sin90 gets turned into 5 · sin90 so that you know that sin and 90 go together and the 5 is multiplied by the result.

To combine the numbers in a different way, enter a different operator between numbers.

i.e. 5 + sin90

This operator is also used if you include multiple results from the "Data" drawers.

You say this program contains bugs. What do you mean by that?
Known, outstanding issues for Magic Number Machine include:

- it could invite other programs over for a party and not clean up the mess
- it could pass messages to the kernel that contain lewd caricatures of you
- it could sign you up for spam
- it could use up all the number 2's on your computer so that you have to start using backwards 5's.

If you experience anything better, consider it a blessing.

What kind of name is Magic Number Machine for a calculator?
Consider the name SGI gave their calculator: SciCalc. Then there was Steven Constenoble's Mac OS 7-9 calculator, SciCalc. Brent Jenkins made a calculator for the Palm OS called SciCalc. There's even a few hundred online javascript calculators called SciCalc. I even wrote a calculator for Mac OS X beta (and subsequently updated it for X 10.1) called... wait for it... SciCalc.

Sure there are some more interesting names: pCalc, MooseCalc, KoalaCalc, MPCalc, etc. The list goes on. I got "Calc" overload.

Instead I turned to the wicked, pagan-sorcery side of arithmetic, creating a device of purely mystical design? Sure, why not. Actually, it's a misheard quote from my childhood (--"I'll trade you three hills and a mountain for this *Magic Number Machine*." -- "Ooooh.").

Your calculator doesn't have [insert feature I don't care about here].
You didn't word your statement in the form of a question. I didn't go through milliseconds of hardship typing a "Q" in the title of this page for no reason.

All right, I'm feeling generous. I'll assume you meant to ask: "How do I get [insert feature I don't care about here] added?" or "Will you add [insert feature I don't care about here] for me?".

The short answers are: you code it yourself and I won't add it for you. The long answer is that if everyone asks for something and it doesn't seem like too much work, then I may add it.

Quick tip: a programmable calculator seems like too much work. You can add your own functions easily with only a little programming and there are other calculators that can do most of the rest (like the HP x48 simulator). A graphing calculator seems like even more work.

Still impatient? Try these:

- MPCalc for really high precision, really quickly
- MooseCalc for dates and times and stuff
- RPN Calculator for unit conversion and complex number stuff
- Mathematica for everything plus a hole in your pocket (seriously: thanks to Wolfram Research for funding Eric Weisstein's ScienceWorld – Magic Number Machine would suck without MathWorld's help).

I can't program, can you help me?
Yes I can, but I won't. Programming is something really hard to teach. It takes years to do well. I will only have time to give minor assistance to would-be developers.

If the back button on your browser is too far away, you can click here to go back to the Help Contents.