iHaxGamez 1.6.25 Documentation

Note: iHaxGamez is for amusement purposes only. By using this program, you accept full responsibility for your actions. You agree that the developers of iHaxGamez are NOT, in ANY way, responsible for any damage to you, your software, your computer, or anything else caused through the use of this or any other program.

Note: Flash applications are special. There is a section at the end about how to hack your favorite Flash based application. Look there now if you know everything else about iHaxGamez. (Even if you think you know about Flash games, I recommend you review the info.)

About iHaxGamez:

iHaxGamez was written by me, Raymond Wilfong, for use on Macintosh OS X. It was designed with two purposes in mind. The first is that Video Games are meant to be fun, and mindless. Extra lives and free money help make the game more fun, and mindless. This program will help you find and adjust these (and many other) values in your game. The second reason is that I wanted to learn Objective-C and the Cocoa framework for Macintosh OS X development. I also wanted to check out the Xcode development tool from Apple.

The current version of iHaxGamez was compiled to be used on OS X 10.5, and newer, on 32 and 64 bit hardware. The disk image for the distribution also has an updated version that works on OS X 10.4. This should only be used with older systems that have not been upgraded to 10.5.

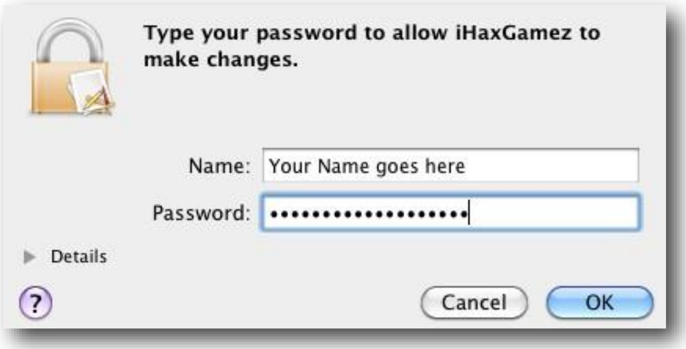
iHaxGamez License:

iHaxGamez is being released as free, open source software under the GPLv2. For more information, contact rwilfong@rewnet.com with “iHaxGamez” in the subject line. Although the license included in the build does not mention this, I am also hereby releasing this program as Public Domain source code, so use it however you want, and don,t worry about license limitations.

iHaxGamez Usage:

iHaxGamez consists of three windows: The Authentication Window, the Main Window, and the Search Window.

The Authentication Window is the standard window that OS X shows when it needs to upgrade the authorization rights of an application. In this case, iHaxGamez needs to be authorized as an administrator of your machine so that it can access the memory of other running applications (such as your game).

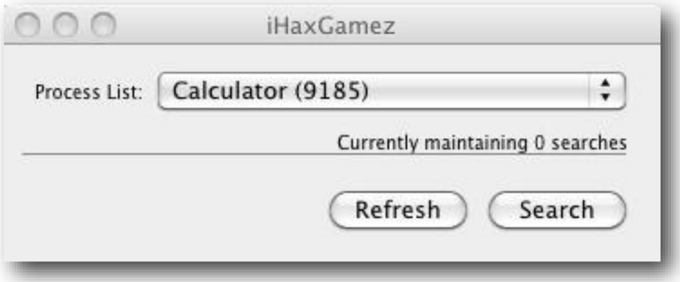


Type your password to allow iHaxGamez to make changes.

Name: Your Name goes here

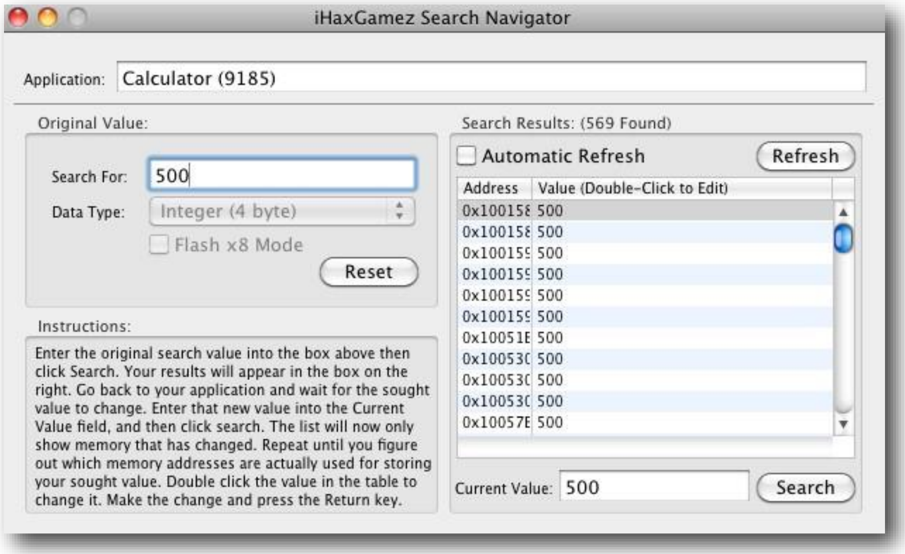
Password: ••••••••••••••••••

The Main Window allows you to select a running process. You may manually refresh the Process List at any time by clicking the Refresh button. The process list refreshes automatically whenever the form is reactivated. Once you have selected a Process, click the Search button to open a Search window for that process.



Process List: Calculator (9185)

Currently maintaining 0 searches



Application: Calculator (9185)

Original Value:

Search For: 500

Data Type: Integer (4 byte)

Flash x8 Mode

Instructions:

Enter the original search value into the box above then click Search. Your results will appear in the box on the right. Go back to your application and wait for the sought value to change. Enter that new value into the Current Value field, and then click search. The list will now only show memory that has changed. Repeat until you figure out which memory addresses are actually used for storing your sought value. Double click the value in the table to change it. Make the change and press the Return key.

Search Results: (569 Found)

Automatic Refresh

Address	Value (Double-Click to Edit)
0x10015E	500
0x10015E	500
0x10015S	500
0x10015S	500
0x10015S	500
0x10015S	500
0x10051E	500
0x10053C	500
0x10053C	500
0x10053C	500
0x10057E	500

Current Value: 500

The Search Window is where most of the work happens. You start by entering a search value into the Search For field. Then you choose the Data Type for the Search as described below.

Choosing the Data Type:

Note: 99.9% of the time you will probably be searching for a 4 Byte integer. This is the default type selected, so leave it unless you understand the following information.

The data type can be a hard decision when you're unfamiliar with the way computers store values. Currently, iHaxGamez can handle searches for Integers (numbers with no fractional part such as 120), Floating-point (Float) numbers (numbers with fractional parts such as 3.14159) and Strings (a collection or characters). You may now be asking why you can chose from four different Integer types, two different Float types, and two types of strings. This is because small Integers need fewer memory locations than big Integers. And, high precision Floating-point numbers need more memory locations than low precision Floating-point numbers. And ASCII character strings take less memory than Unicode strings.

When the original programmer of your game wanted to store the number of lives you can have, he or she decided what size Integer you would need. Most of the time, it's easiest to use a 4 byte (32 bit) Integer. But sometimes, to save computer memory, they decide to only use two or one byte Integers. If the largest Integer value will be 127 or less, one byte will work. If 32767 is the limit, two bytes are needed. For Billions (2,147,483,647) you need 4 bytes. Note: For those of you who know all this, I decided not to make the user decide between signed and unsigned integers. All entered integers are presumed to be unsigned. If you enter a negative integer it will be displayed back as a big positive number.

Floating-point numbers work in a similar way. However, the rules for deciding which to use are different because Floating-point numbers can be very large, but the size difference (4 bytes or 8 bytes) is based on the number of significant digits. For example, 3.14 has three significant digits and 3.1415927 has eight significant digits. I suggest you search for 8 byte floats first because you will get fewer results than if you search for 4 byte floats. If nothing comes back, or you cannot find your desired memory location, try using 4 byte floats.

Strings can have multiple sizes too. In ancient days of computer science (before the mid-1990s), computers used ASCII codes, and extended ASCII codes, to store characters. Each character was stored in one byte of memory. Today, because computers communicate in multiple languages, and therefore have more than 256 letters/digits/symbols, a new standard had to be created. It's called Unicode. Strings in programs may be stored in both sets within the same program, so good luck figuring out which one is best for your search. I suggest you try Unicode first, and if you can't find what you want, try the ASCII.

How to Search and Replace:

So, you've entered a value and selected a type. Now you click the Search button. The search button will magically become a Reset button, and iHaxGamez will begin searching the memory of your Process. When completed, the Search Results pane will contain a list of all memory addresses that have the value you were looking for. This list will probably be longer than you expected. How do you know which address is the one you're looking for?

The short answer is, you wait until the value changes. Then you Search through all the original addresses, which contained your original value, for the new value. This is a process of elimination. After a few iterations, you should have found your address. If two or three addresses are left, after two or three iterations, you probably need to change all the addresses to your new value. This happens because the programmer decided that the value is needed by multiple parts of the program.

Here's an example: You start a game with 20,000 coins. You pause your game and run iHaxGamez. Now you search for a 4 byte integer containing the value 20000. When the search completes, the window will contain a large list of addresses that contained the value, 20000. Now, you go back to your game and spend 25 coins. Then, pause your game, and return to iHaxGamez. Enter 19975 in the Current Value field, and click Search. iHaxGamez will look at only the addresses found previously. When the list has an address that does not contain your new value, the address is removed from the list. In theory, you will have a much shorter list. Repeat this process as needed. Once you're down to one or two memory addresses that always change to your new amount, you can start changing values.

How to Change a Value:

Double-click on the value in the list, and it will be in edit mode. Type the new value and hit the Return key. Presto: you're RICH!!! (NOTE: Don't forget about the maximum values for 1 Byte, 2 Byte and 4 Byte Integers as explained above. Entering larger numbers may give you negative amounts in your game.)

There's one thing to keep in mind when working with strings. Because I believe in safety, when replacing strings, the replacement string CANNOT be longer than the original search string. Attempts to place a longer string into the list's value will result in a truncated version of your new string being stored. I considered this necessary because if you write to memory beyond the limits of the string length, you could corrupt the program, and who knows what could happen next. I know what you're thinking: "iHaxGamez is inherently dangerous because it's writing to memory that it does not control, and that is evil!” This is true, but I try to stay away from buffer overflows. If you don’t like it, edit the source code, and recompile your own version.

About Flash games:

Most of the questions I get involve hacking Flash based games. I decided to throw a few facts together to help you along. First of all, Flash often (NOT ALWAYS, JUST OFTEN) stores its values as eight times the actual integer amount. So, if you’re looking for the value, 5, you used to have to search for 40. When you wanted to replace that value, you used to have to enter eight times the replacement value (IE: 800 to store the value 100).

With version 1.6.1 of iHaxGamez, there is a check box under the initial search value that says, “Flash x8 Mode”. If this is checked, the multiplication is done for you.

My second fact concerning Flash is something I learned when I upgraded to OS X 10.6, Snow Leopard. When Flash based applications run, the process you must connect to is NOT Safari. In the past you would connect to the web browser that you were using to access the Flash application. However, Safari now uses another process called “WebKitPluginHost”. I believe this was done to protect the OS from “evil” Flash applications. It puts the Flash application in a cage that is harder to get out of than just a web browser. Keep this in mind when you are trying to hack your next flash game.