
Casio-BASIC:
A guide for TI-BASIC Programmers
By VeninGriffe

Introduction

The general purpose of this guide is to help people who currently program in any of the variants of BASIC found on Texas Instruments' Graphing calculators transition to the BASIC present on Casio Calculators, most specifically the 9860G series, although other series will be mentioned. This specificity is not due to any particular preference on my part, but is largely the result of the availability of information on this particular series. In contrast to the rather thin User's Guide provided by Texas Instruments to their Users, Casio has made a six hundred page User's Guide available that is specifically directed at using the software on the calculators. Much of the information in this tutorial comes from that guide, which can be found on Casio's website under the title “fx-9860 G Series User's Guide.”

While the tutorial is geared towards one series, the Casio-BASIC found on all series introduced after 1995 (second generation) is very similar. The only significant exception to this rule is the 1996 fx-7400 series, which uses the first generation variant of Casio-BASIC. This guide will deal exclusively with the second generation language, as the first generation version is so limited as to be almost unusable for any non-trivial programs.

Formatting

Throughout the course of this guide, I will attempt to make formatting as consistent and logical as possible to aid in readability. The syntax of Casio-BASIC often resembles that of TI-BASIC, so confusion would otherwise be likely. Indeed, the first time I looked at a Casio-BASIC program, it took me a

second to realize that it wasn't actually TI-BASIC. To help avoid this, code in TI-BASIC and code in Casio-BASIC will hereafter be differentiated like this:

```
: .TI //comments will be made like this
: Disp "Hello World" //data inside quotation marks will appear italicized
```

```
: .Casio
: Locate 1,1,"Hello World" //see how the syntax can be different?
```

In other words, Dark Grey code means that the code is written in TI-BASIC. while light Grey means that the code is Casio-BASIC. Also, the colons are used in the Casio-BASIC code for readability, even though they are not present in the on-calc programming environment.

Finally, onto the tutorial

Now that we've made it through the introduction, we can get to the tutorial. Since the goal of this tutorial is to help you learn Casio-BASIC from the foundations of TI-BASIC, I will assume that you have some rudimentary knowledge of the commands. But, before we get into programming, you'll have to figure out how to start a new program. The process is fairly simple and isn't much different from that of Texas Instruments calculators. First, enter the Program menu by pressing “Shift” and then “Vars.” Note that this may change for some calculators. After you've opened the menu, press the F3 button. A menu will appear prompting you to choose a name for the program. The name must be a maximum of eight characters long and should contain only the characters A-Z, r, θ, and spaces. You can use other characters, but try to stay away from them for the sake of readability. Keep in mind that no matter how long or short your program name is, it will automatically use 32 bytes of memory. Now comes the first major difference between Casio-BASIC and TI-BASIC. With Casio-BASIC, you can lock the source code from viewing through the use of a password. A password locked program can still be executed, but the source

code cannot be viewed or edited without the password. As useful as this is, it has one drawback for programmers: Password locked programs cannot be debugged with the built in debugger. However, when the calculator asks you if you want to register a password after you enter the name (press F5), accept it. You cannot go back and put a password lock in later. TI calculators don't have a debugger anyway, so you shouldn't miss it. Now that you can start programs, we can start with the actual programming.

Text Commands

Let's take a look at the code from before:

```
: .TI
: Disp  Hello World
```

Seeing as you are already at least a rudimentary TI-BASIC programmer, you should recognize that this routine will output the string “Hello World” to the home screen. However, the Casio-BASIC routine looks different.

```
: .Casio
: Locate 1,1, Hello World
```

What exactly is this “Locate” command? Put simply, it's the Casio-BASIC version of the Output(command. The arguments for this token are

```
: Locate x,y, Hello World
Or
: Locate x, y, expr
```

Yes, just like Output(, Locate will accept an expression as well as a string. And, exactly as Output(does, Locate will compute [and display] the value of the expression. The x and y values are also similar to the arguments of Output(. They stand for the row and column in which the first character of the string or computed expression will be displayed. In the 9860 G series, the home screen is 21 characters wide and 7 characters high. Thus, your x values will be integers between 1 and 21 while your y values will be integers between 1 and 7. Decimal numbers will be rounded to the nearest integer. The program will return a syntax

error if the x or y values are outside of these ranges. Also, if your text string is more than 21 characters wide, the extra characters will not be displayed.

You might have noticed how many comparisons I've made between Locate and Output(. But, the original TI-BASIC code used Disp. Why didn't it use Output(instead? There really isn't a reason it was written like that. However, Casio-BASIC does have a command that's more similar to Disp.

```
: "Hello World"
```

Simply printing the string within quotation marks is essentially equivalent to Disp. The command is simple enough that it doesn't need any further explanation and like Disp, you probably won't want to use it in any programs that you plan on releasing.

In TI-BASIC, there are commands other than Disp and Output(, though. Most notably, you can use the Text(command to draw text to the graph screen. Casio-BASIC also has an equivalent command, which is conveniently called Text.

```
: Text y, x, "Hello World"
Or
: Text y, x, expr
```

There is one important difference between the TI-BASIC Text(and the Casio-BASIC Text: The Casio version does not update the graph screen when it is drawn. You must use another command such as Plot to update the screen. This was done to increase the speed of the command. Also, just like Locate, if the string of characters is longer than the screen can hold, the extra characters will not roll onto another line. They will simply not be displayed. Further, according to one source, Casio reported that only certain tokens could be used with the Text command in the manual. As far as I can tell, there are no restrictions on tokens with the Text command. Before we move onto new commands, there is one final thing that should be mentioned. The x and y arguments for Text are not the same as those for Locate. They are similar to the

x and y arguments for the TI-BASIC Text(. In other words, x and y are the number of pixels from the top left corner of the graph screen.

There is one more text command: the output command, which is represented by a triangle similar to ▲. It's this symbol that will be used to represent the command throughout the rest of this tutorial. To use output, simply place ▲ after any computation in the program. For example:

```
: .Casio
: 1→A
: A+1▲
```

In all honesty, the ▲ command is next to useless for all text display other than developmental debugging. When the above routine executes, you will see the string “-Disp-” where the data will be displayed. After you press EXE, the string will change to the data the ▲ command will display. It takes yet another EXE keypress to return to the normal program flow. It does have other uses that make up partially for its complete uselessness, as we'll get to later in the tutorial.

Input/Output Commands

As nice as it is to display text, sometimes you want the program to interact with the user. With a Texas Instruments calculator, you might use something like Prompt or Input to get data from the user. For example, the code

```
: .TI
: Prompt A
```

might be used to get input from the user and would display

Casio-BASIC has a similar command, officially known as Input. The input command is designated by the ? Token. To use the command, “store” the ?

Token to a variable.

```
: Casio
: ?→A //notice that the syntax is like storing a number to the variable
```

The program will output

Note that although the above screen shot is a replication using TI-BASIC and WabbitEmu, this is generally how the routine will appear on your Casio. Random question marks are great, but how does the user know what to input? Basically, how can we attach text to the input routine? In TI-BASIC, one might use something like:

```
: .TI
: Input "Hello World",A
```

In Casio-BASIC, this is really quite simple. All we need to do is place a text string before the input command:

```
: .Casio
: "Hello World"?→A
```

This will display a screen similar to

Remember that the quotation marks only go around the string. The ? Token does not go inside them. If the token is inside of them, the interpreter will not register the token as a command, but instead as part of the string. Another feature of the input command is that if the user gives bad input, such as text, the

command will automatically return a syntax error. If the same user then presses the left arrow, it will allow them to re-enter the data correctly.

For text based programs, a textual input command is useful. In many cases however, what you want is key input. In such situations, Input is utterly useless. Luckily, Casio put enough thought into Casio-BASIC to include a GetKey function. The GetKey function can be implemented as

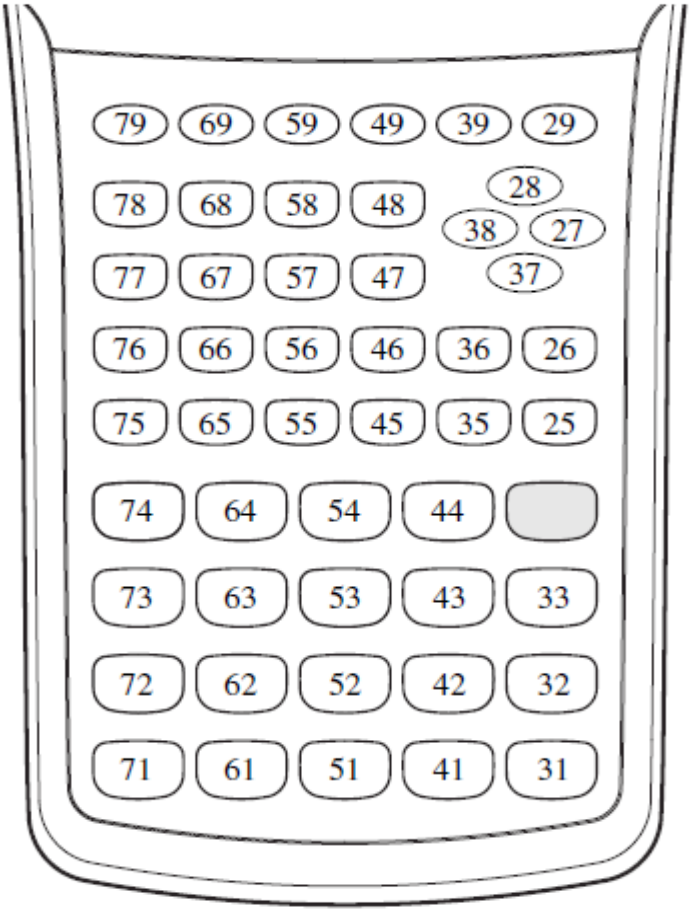
```
: .Casio
: GetKey→A
```

Getkey records the value of the last key pressed and returns the value of that key. In the above code, that value is stored to the variable A. However, GetKey can be used as an argument by itself. For example:

```
: .Casio
: While Not GetKey
:  WhileEnd
```

Since the value of GetKey changes every time a key is pressed, storing its value to a variable gives you control over the input and leads to consistent results from computations. This syntax does have its uses, but just store GetKey to a variable for now.

While we're on the subject of the GetKey function, it might be rather prudent to give you the key codes returned by the command. For the fx-9860 series, the keys are:



Like you, I have no idea what happened to the keys below 25. My best guess is that the OS uses those to address components within the hardware of the calculator itself.

There are only a few more I/O commands. Collectively, you can call these the data transfer commands. Just as you might imagine, these are used to send or receive information from other calculators. For whatever reason, Casio decided that more commands were better and included six different commands for sending or receiving data: Send(, Receive(, Opencomport38k, Closecomport38K, Send38K, and Receive38K. I believe the latter four are hardware specific and may not be included on all calculators, but the first two are almost certainly present no matter what second generation calculator you own. These require data structures to operate, so they'll be addressed later in this tutorial. Until then, you'll just have to refrain from making multi-calc games.