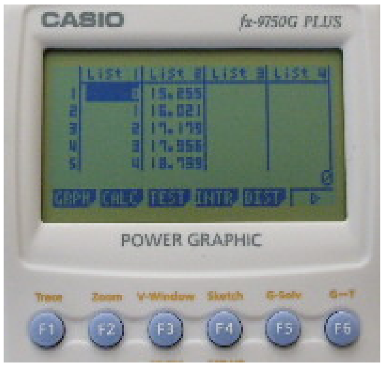
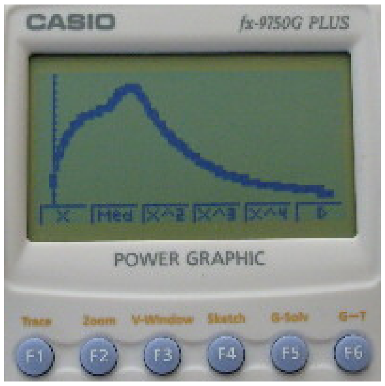
Connecting the PICAXE 08M and PICAXE 18X to the Casio 9750G Plus graphics calculator

Introduction to Casio BASIC





Compiled from various web resources
July 2008 © Michael Fenton

Casio BASIC

PART 1: Creating a Program

To create a new program press MENU then locate the program icon, press EXE, press F3 to create a new program. You will then be asked to enter a name for your program, also you may create a password at this time (WARNING: if you put a password on your program you will not be able to use the debugger).

PART 2: Variable Basics

A variable is something which **holds a value**. The most commonly used are the letters A~Z and r,_. There are others, but they are not covered in this section. To store a value into a variable you must use the → key, like this:

CODE

```
1->A  // Assigns 1 to A
```

It should look like this:

CODE

```
1->A~D // Assigns 1 to A B C & D
```

Variables are the most important aspect of programming.

PART 3: Basic Loops

Loops are used to **repeat** blocks of code. There are several types of loops in Casio BASIC: Goto/Lbl, Do/lpWhle, For/Next, and While/WhileEnd. Here is an example of a Goto/Lbl loop:

CODE

```
1->A  //Assigns 1 to A
Lbl 1
A+1->A  //Adds 1 to the variable A and stores new value
Goto 1  //Returns to Lbl 1
```

This loop will execute infinitely, Adding 1 to A each time (actually it will stop when the value of A is greater than 99*10^99 because you will get a mem error).

Here is an example of a *For* loop; note that it does not take as many lines:

CODE

```
For 1->A to 100 Step 1
//this loop assigns 1 to A then adds 1 to A until it equals 100
Next //Goes back to the start of the loop, adds the step to the
variable.
```

Anything between the *For* and *next* statement will be executed until the expression evaluates true (A equals 100). By changing the value of *Step* you can change how much A is incremented by.

The *While* loop checks to see if the expression is true then executes the code. After the code has been executed it returns to the top, checks the expression, and if it is false jumps out of the loop and continues with executing the program. This is an example of a While loop.

CODE

```
1->A
While 1=1 //1 always equals 1 so the expression always evaluates to
True
A+1->A
WhileEnd
```

Since the while loop evaluates the expression before executing the code it is possible that if the expression is false before loop begins (ex: 1=2) then the loop will never occur, it will just skip right over the code and continue with the program.

Unlike a while loop a Do/LpWhle loop will always execute at least once since it evaluates the expression after the code has been executed. A Do/LpWhle loop looks like this.

CODE

```
1->A
Do   //Start of the Do/LpWhle loop
A+1->A
LpWhle A<100 //Loops while A is less than 100
```

PART 4: Selection Statements

Selection Statements are used to make programs have differing outcomes, instead to executing the same way every time they are run. A selection statement checks an expression, sees if it is True or False, then if True executes the rest of the statement otherwise it skips to below the statement and continues with execution. There are two types of selection statements on the calculator. They are the If/Else statement and the => arrow.

An *If/Else* statement works like this

CODE

```
1->A
If A=1   //expression to be evaluated
Then "HI"  //result if expression is true
"HOW ARE YOU"
Else "BYE"  //result if expression is false
"SEE YOU LATER"
If End   //end of statement
```

Result:

Since A is 1 the statement evaluates to true, therefore

QUOTE (Program Output)

```
HI
HOW ARE YOU
```

is printed. If you replaced the first line with 0->A then the if statement would evaluate to False, and

QUOTE (Program output)

```
BYE
SEE YOU LATER
```

would be printed.

An If statement can contain many different things, and can be many lines long, they are the keystone to making a game, and before going on you should feel comfortable using them.
The => arrow is also very useful, it is a single line selection statement that takes up less space but can do less than a normal if/else.

This is the same code as above, except using the => arrow instead of the *If* statement.

CODE

```
1->A
A=1=>"HI"
A=1=>"HOW ARE YOU"
```

```
A<>1=>"BYE"   // <> means not equal to
A<>1=>"SEE YOU LATER"
```

In general, the best time to use a if statement is when you have at least two lines of code to be put inside, otherwise use the => arrow.

PART 5: Advanced Loops

Nesting is when you take one loop and put it inside another. Here is an example:

CODE

```
0->C
For 1->A To 10  //Step can be omitted instead of using Step 1
For 1->B To 10
A+B+C->C
Next
Next
C◢   // the ◢ represents the output sign, it displays whatever is
before it and Pauses until [EXE] is pressed.
```

By executing this code you will get this output:

QUOTE (Program Output)

```
1100  //caused by ◢
```

Now we will go through the code and look at what each line does.

Line 1: 0->C

Assigns the value 0 to the variable C

Line 2: For 1->A To 10

Tells the program that you will be looping until A is equal to 10, adding 1 with each loop.

Line 3: For 1->B To 10

Tells the program that you will be looping until B is equal to 10, adding 1 with each loop.

Line 4: A+B+C->C

The current value in A will be added to the current value of B, that value is then added to the current value of C and then assigned to C.

Line 5: Next

Goes to line 3.

Line 6: Next

Goes to line 2.

Line 7: C◢

Displays the final value of C.

As you can see, the second loop executes completely for each iteration (loop) of the first loop.

Nesting also applies to selection statements, you can nest if or => inside each other as needed.

Example:

CODE

```
A<1=>A>0=>"HI"
If A<1
Then if A>0
Then "HI"
End If
End If
```

<u>Part 6: Logical Operators</u>

As you have just seen, there are times when you will want more than one condition in a selection or loop, you can nest the statements or you can use a logical operator: AND OR, NOT.

AND and OR are operators which can be used in selection statements to specify additional conditions, therefore the code above could be written:

CODE

```
A<1 and A>0=>"HI"
```

Instead of:

CODE

```
A<1=>A>0=>"HI"
```

The OR operator allows you to specify alternate conditions which are evaluated independently and if any are true then the entire statement is considered true. This means you can do this:

CODE

```
If A=1 Or B=1
Then "TRUE"
End If
```

When this executes it will print TRUE if A or B equals 1.

The NOT operator can also be used in selection statements, though it is not used nearly as often. NOT returns False if the statement is true (or 1) and True if the statement is False (or 0) so:

CODE

```
If Not (A=1)
Then "TRUE"
End If
```

This will print true if A is any number but 1. In this case the Not statement is used as an <> (not equal to) statement, but it has other uses. We will cover more on this later.

Part 7: Input Basics

You might want to get information from the user. The simplest way to get input from the user is to use the ? command.

Syntax: ?->(Variable)
Description: this command causes a ? to appear on the screen, execution pauses until the user enters data and hits EXE, then the data is put into the variable. This is a good way to ask yes or no questions, with the user having to enter 1 for yes and 0 for no.

Another way you can use the ? command is to put a prompt before it: "Continue (1=YES 0=NO)"?->C
This causes the text to be displayed with a ? after it and waits for data.

Part 8: Key Input

Sometimes you will want to know what key the user is pressing, to do this you must use the Getkey command.

Syntax: Getkey->(variable) or if Getkey=(key number)

Description: Getkey returns a different value for each key pressed, if no key is pressed then it returns 0, here is a simple program used to display the value of a key.

CODE

```
Lbl 1
Locate 1,1,Getkey
Goto 1
```

The only key which does not return a value is AC/On.

Part 9: Advanced Loops

A simple loop:

CODE

```
While A<>B
WhileEnd
```

A nested loop:

CODE

```
While A<>B
Do
LpWhile A=B
WhileEnd
```

A nested loop is just a loop inside of a loop, just like a nested if. what most people have trouble with is the execution of nested loops, if you have programming experience then you will all ready know this but if your new to field take a look at the sample below:

CODE

```
Lbl 1
1->A
1->B
1->C //counter var
1->D //counter var
Do
A+B->A
While D<=5 //loops 5 times
B+A->B
Isz D
WhileEnd
1->D
Isz C
LpWhile C<3 //loops 3 times
A+B◢
```

Go through this program and see if you can figure out what the output will be, then look below for the answer.

if you did not get 610 as the output then you made a common mistake. You do not understand one of the basics of nested loops:

QUOTE

PRIMARY RULE: an inner loop executes fully for each interation (loop) of an outer loop

The break statement: Break terminates execution of a loop and resumes normal program flow at the end of that loop.

CODE

```
While A=A //endless loop
Isz A
Break
"THIS ISN'T SHOWN, IT ISN'T EVEN LOOKED AT BY THE PROGRAM"
"NEITHER IS THIS"
WhileEnd
"ESCAPE FROM THE LOOP" //Break goes to here
```

now we will look at how Break works in nested loops

CODE

```
While A<>B
Do
Break
LpWhile A=A
//Break goes to here
WhileEnd //loops like normal
```

As you can see, Break only exits the current loop, and when that loop comes around again and if Break isn't executed (if it's in an If statement) then the loop does not terminate and continues like normal. Get it?

CASIO Basic examples

Example 1:

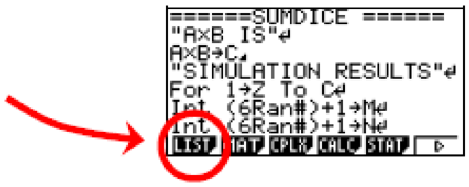
```
====SUMDICE===
Lbl 1
"First number"? -> A
If Int (A) ≠  A
Then Goto 1
IfEnd
"Second number"? -> B
AxB◢
Goto 1
```

How It Works
We have told the program to take the integer of A and test it against the actual value of A. If these two don't equal, then obviously A is not an integer. The next part of the statement is the Then command, since we said, 'if they don't equal, then go to label 1'. If A is an integer, the program will ignore the Then statement.
The ⇑ symbol is in the **PRGM** (**SHIFT** VARS) menu and it means 'stop the program and display this result'

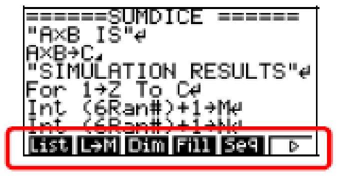
Example 2:
Storing data in a list

The usefulness of the results in the simulation is limited because the outcomes are not stored anywhere, since the variable S is always changing during each loop. This is why it is a good idea to store data in lists (sometimes called arrays), where it can then be analysed later. This is the purpose of this lesson.

In order to store the data in a list, we must first set the dimensions of the list. This means telling the calculator how many elements there are to be stored.
Since we are using the variable C as the number of simulations, this will be used as the dimensions of list 1.
Lets set C equal to 5….
`C -> 5`
Then add the line **`C -> Dim List 1`**
The commands Dim and List are found in the LIST menu, which is in the OPTN menu. Press OPTN, followed by LIST (F1). Just about any action that relates to lists will be found in this menu.

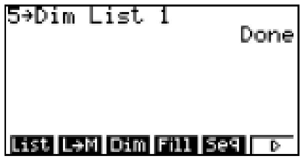


```
======SUMDICE ======
"A×B IS"↵
A×B→C◢
"SIMULATION RESULTS"↵
For 1→Z To C↵
Int (6Ran#)+1→M↵
Int (6Ran#)+1→N↵
```

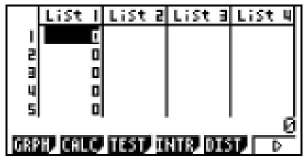


```
======SUMDICE ======
"A×B IS"↵
A×B→C◢
"SIMULATION RESULTS"↵
For 1→Z To C↵
Int (6Ran#)+1→M↵
```

A Bit of Extra Help
Enter RUN *mode from the* Main Menu*. Enter this line and press* **EXE***.*
`C -> Dim List 1`
This line says that I want to have 5 'empty' elements in List 1 (they are in fact filled with the value of 0).
Press MENU *and then enter* STAT *mode. You will see that there are 5 'empty' elements in List 1.*



```
5→Dim List 1
                 Done
```



```
   List 1  List 2  List 3  List 4
1       0
2       0
3       0
4       0
5       0
                                0
```

You will notice that this command has created 5 'empty' elements in list 1. As a rule, whenever you want data to be stored in a list (during the execution of a program), you must first create empty elements using the command described above. This is, if you tried to enter data in position 6, you would receive an error message.

CODE

```
For 1 -> Z to C
Int (6Ran#)+1 -> M
Int (6Ran#)+1 -> N
M + N -> S◢
S -> List 1[Z]
Next
```

How It Works

When the program first comes to the For statement, it will set Z to 1 and go through and execute everything on the way to the Next command. This means it will get to **`S -> List 1[Z]`**
and send the value of S (which has been set as M+N) to List 1, row Z, which at the moment is 1. After this, the program will move on to the next line, which is the Next command in this case.
Once it gets to the Next command, the program will go back to the For command and do it again, except Z will now be set to 2. So then the new value of S will be sent to List 1, row Z, which is now 2. This will continue over and over again until Z=C.
Notice how having the **`S -> List 1[Z]`**
command within the For statement utilises the way Z increases by 1 each time.

The maximum number of elements that this particular calculator can handle in 255.

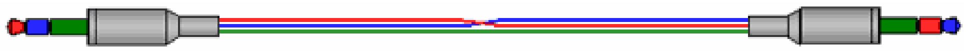
PICAXE circuits

1. Electrical communications:
TTL-level (High=5V and Low=0V), asynchronous serial half-duplex communication, connects to a PC´s RS-232 COM-port with a MAX232 or equivalent level-converter circuit.

- Baud-rate: 9600 bps
- Parity: none
- Byte-size: 8 bits
- Stop-bits: FROM Casio: 2 bits TO Casio: 1 bit

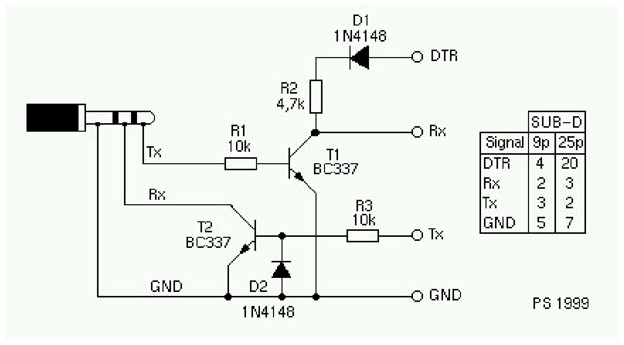
The Casio-plug is a standard 2.5mm stereo jack with the following pin-designations:
- Sleeve Ground
- Ring Data to Casio
- Tip Data from Casio

2. Casio to Casio cross-over cable



2.5 mm stereo plug 2.5 mm stereo plug

3. Casio to PC RS232 interface



	SUB-D	
Signal	9p	25p
DTR	4	20
Rx	2	3
Tx	3	2
GND	5	7

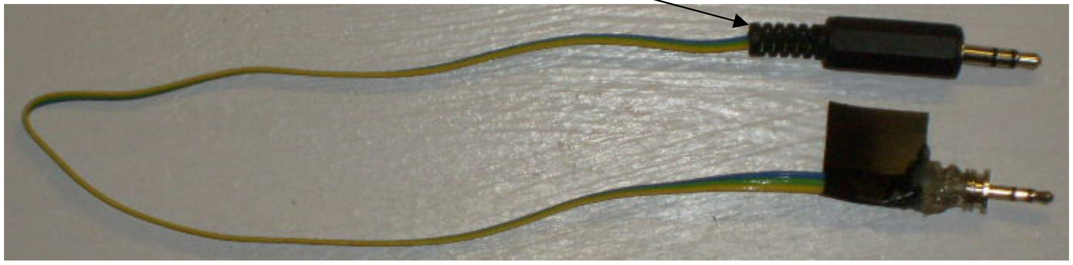
2.5 mm stereo plug from Casio connected to a PC D9 female plug.

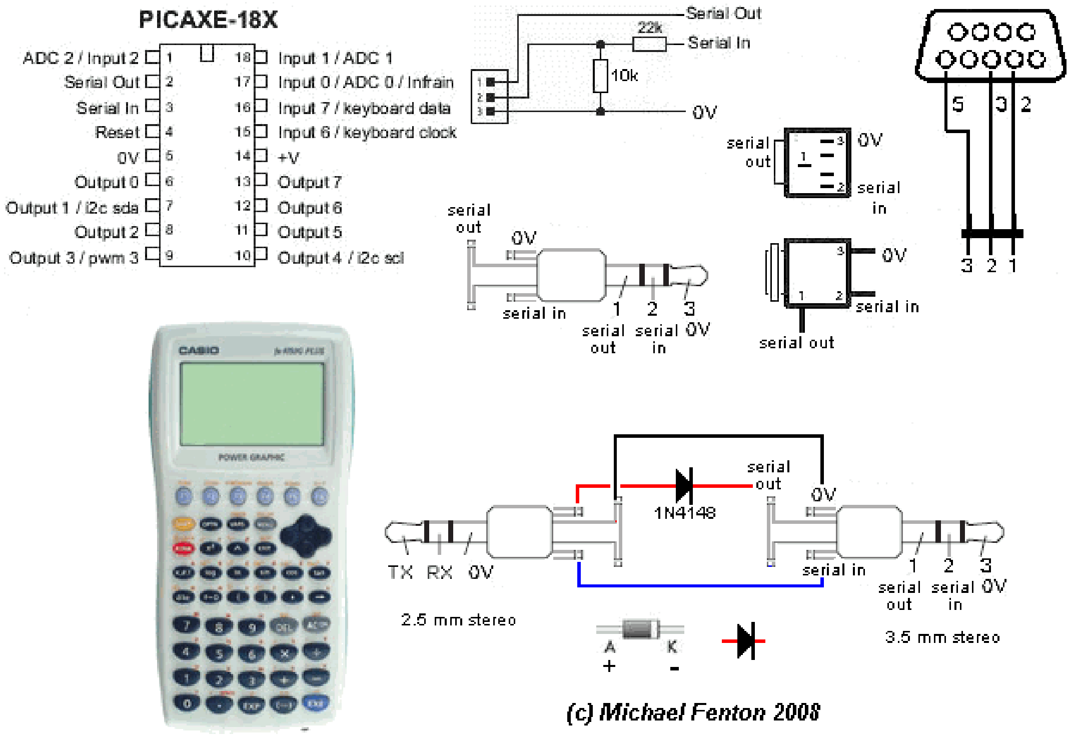
BC547C can substitute for BC337 NPN transistors.

LED’s in series with Rx and TX can indicate communication to and from PC

4. Casio to Picaxe

Place small signal diode 1N4148 (bar toward Picaxe) in series with output TO Casio inside the 3.5mm stereo plug.





6. Casio 9750G Plus Code: RECEIVE picaxe data into a List:

```
255 -> Dim List 1
255 -> Dim List 2
1 -> B
Lbl 1
"Get readings (1 = YES)"?  -> A
A = 1 => Goto 2
Goto 1
Lbl 2
Receive (R)
// R x 0.123 -> A   convert to degree celcius or whatever units
Getkey -> C
If C ≠1 47
Then Goto 3
IfEnd
If B < 256
Then B-1 -> List1[B]
A -> List2[B]
B+1 -> B
IfEnd
Lbl 3
ClrText
Locate 1,1, B-1
Locate 6,1, A
If B = 256
Then Locate 1,2, "LIST FULL!"
IfEnd

Goto 2
```

How It Works

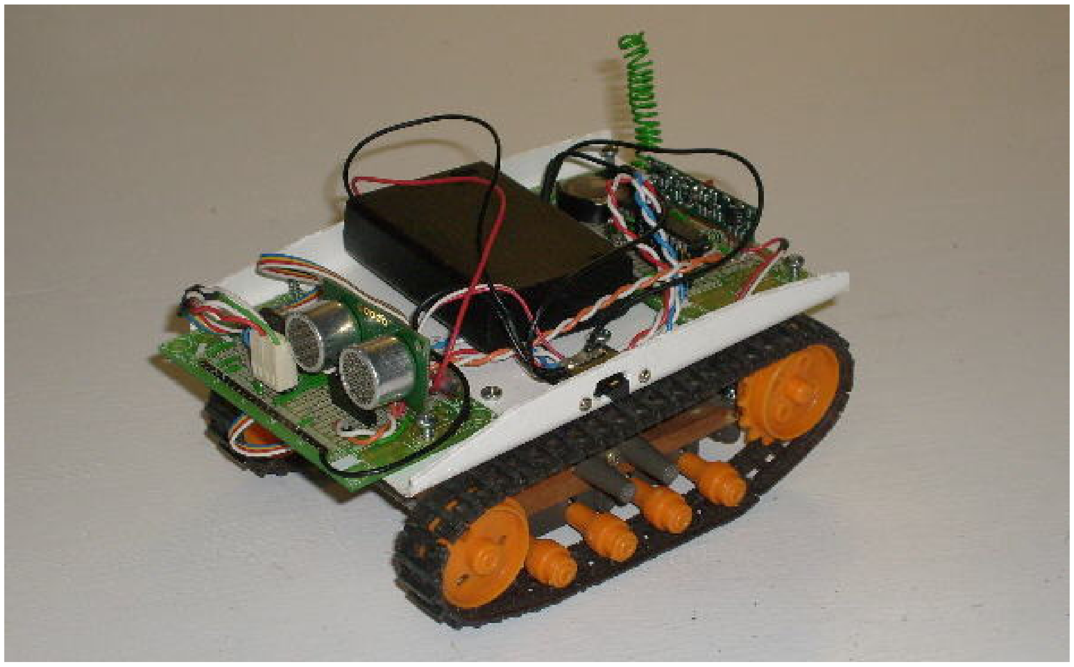
- List 1 and List 2 is created with 255 spaces (rows) assigned to them
- Once the user inputs "1" the Casio displays data values from the PICAXE but only puts them into LIST 2 (y axis) if the EXIT button is pressed. List 1 is for plotting the x axis 0 – 255 time intervals.
- From the MENU select STATs (2 on main menu)
- press F1 (GRPH) then F6 (SET) to set up the graphing options.
- Go down to graph type and press F6 to get to other graph types then choose box (for a box and whisker).
- Press EXIT back to the list (you should be able to see your data in the list), press F1 to get a box and whisker graph, good for working out mean, upper quartile and range.

The data collected can be transformed to temperature, etc by adding line in blue above (remove // remark symbol)

Students can try to fit a x^2 or other curve and explain

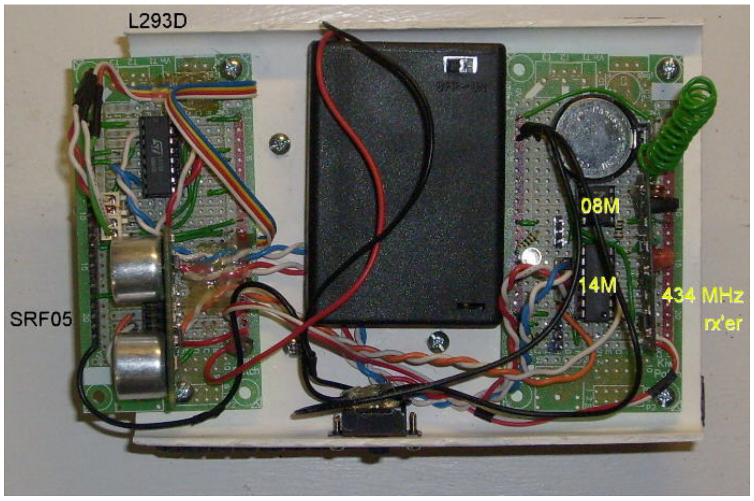
7. Casio 9750G Plus Code: SEND keystrokes to picaxe ypod to control *CASI*

```
Lbl 1
Getkey -> A
If A  > 0
Then Int((A-26); 2.5 +0.2) -> A  //  ;  means divided by
Send (A)
IfEnd
Goto 1
```



How It Works

- *CASI* has an 08M slave that listens for Casio keystrokes
- *CASI* 08M interrupts a 14M master with the keystroke
- *CASI* turns on a L293D motor controller chip to move CASI in the appropriate direction
- SRF05 sonar range finder prevents obstacle collisions



8. Casio 9750G packet encoding:

Modified from article by Erik Grindheim, August ~ October 2001

There are **four** different types of data packets in use to transfer variables, in addition to the single-byte packets. This section describes the structure of these packets in detail:

<u>Request packet:</u>

Byte no (1-50)	ASCII	HEX ($)
1-4	:REQ	$3A $52 $45 $51
5		$00
6-7	VM (Variable) PC (Picture) LT (List) MT (Matrix)	$56 $4D <-- This is the bytes we want... $50 $43 (Byte no 12-49 below is based) $4C $54 (on Variable transfers, VM.) $4D $54 ( )
8-11		all $FF
12	A-Z / r / è	(The name of the Variable:) $41 - $5A / $CD / $CE
13-49		all $FF
50		Checksum = $01 + not((sum bytes 1-49)- $3A)

<u>Variable description packet:</u>

Byte no (1-50)	ASCII	HEX ($)
1-4	:VAL	$3A $56 $41 $4C
5		$00
6-7	VM	$56 $4D
8		$00
9		if the variable is reset/unused: $00 ** if variable is in use (normal): $01
10		$00
11		the same value as byte no 9: $00 or $01
12	A-Z / r / è	
13-19		all $FF
20-27	Variable	$56 $61 $72 $69 $61 $62 $6C $65
28	C or R	$43 or $52 (If the variable has an imaginary part: C-complex. Otherwise: R-real)
29		$0A
30-49		all $FF
50		Checksum = $01 + not((sum bytes 1-49)- $3A)
		**note: If bytes 9 and 11 has the HEX-value 00 this means that the variable has not been used after last "Alpha Memory" reset. Then no Value packet will be transmitted at all. After this packet comes the End packet.

Variable packet, Real numbers:

Byte no (1-50)	ASCII	HEX ($)
1	:	$3A
2-5		$00 $01 $00 $01
6		$00-$09 (BCD 0 and BCD 0-9 (integer part))
7-13		$00-$99 (BCD for 14 digits (decimal part))
14		SignInfoByte. Bit set if...: bit 7 - Variable has an imaginary part bit 6 & 4 - Value (real part) is negative bit 0 - Absolute value of (real part of) value is 1,0... or more bits 1, 2, 3 and 5 are always 0 / Low
15		Exponent, 00-99 Exp.: Byte: +99 99 \\ +01 01 -} byte 14 bit 0 = 1 (High) ** 00 00 / -01 99 \\ -02 98 -} byte 14 bit 0 = 0 (Low) -99 01 /
16		Checksum = $01 + not((sum bytes 1-15)- $3A)
		**note: If the variable is (exactly) 0,0 then the exponent is 00 (instead of minus infinite) but this bit (bit 0 in byte 14) is still low because variable value is less than 1.

The smallest value the Casio accepts is ± 1.00000000000000 × 10 $^{-99}$
The greatest number the Casio accepts is ± 9.99999999999999 × 10 $^{+99}$
Zero is stored as + 0.00000000000000 × 10 $^{+00}$

Or, at a general form: ± I.decimals × 10 $^{+EE}$

In the *Value packet*, byte no. 6 is containing the integer part I, always from $01 to $09 (except if the variable value is exactly 0,0…).
Bytes no. 7 to 13 contain the 14 decimals represented as BCD data.
The exponent EE is represented in byte no. 15 and bit 0 of the SignInfoByte (byte no. 14).
The sign indicating whether the value is negative or positive is held in bits 6 and 4 of the SignInfoByte.

Variable packet, Complex numbers:

Byte no (1-50)	ASCII	HEX ($)
1-15		Equal to bytes 1-15 in the 16 bytes long Value packet for real numbers. These bytes only describe the real part of the variable, while the next 10 bytes describe the imaginary part of the complex number:
16		$00-$09 (BCD 0 and BCD 0-9 (integer part))
17-23		$00-$99 (BCD for 14 digits (decimal part))
24		SignInfoByte. Bit set if...: bit 7 - Always set to 1 (imaginary) bit 6&4 - Value (img. part) is negative bit 0 - Absolute value of (img. part of) value is 1,0... or more bits 1, 2, 3 and 5 are always 0 / Low
25		Exponent for the imaginary part; 00-99 Exp.: Byte: +99 99 \\ +01 01 -} byte 24 bit 0 = 1 (High) 00 00 / -01 99 \\ -02 98 -} byte 24 bit 0 = 0 (Low) -99 01 /
26		Checksum = $01 + not((sum bytes 1-25)- $3A)

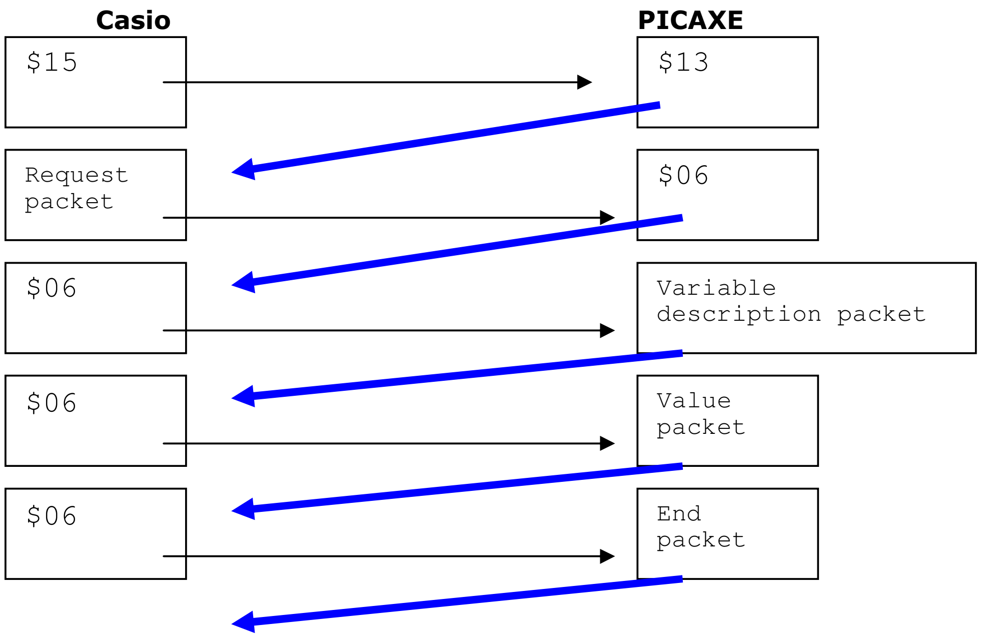
End packet (aways the same!):

Byte no (1-50)	ASCII	HEX ($)
1-4	:END	$3A $45 $4E $44
5-49		all $FF
50	V	$56 (checksum, always the same value for all End packets no matter what other packets have been sent)

Checksum calculation:
As an example let's show how the checksum of the *End packet* is calculated. This value will always remain the same since nothing in this packet changes from time to time. First we add all the previous 49 bytes. This gives us a sum of $E4 (the excess carry digits are thrown away). Then we subtract $3A and the result is $AA. We invert it (=$55) and add $01. The result is $56, which is the checksum-byte; byte no. 50.

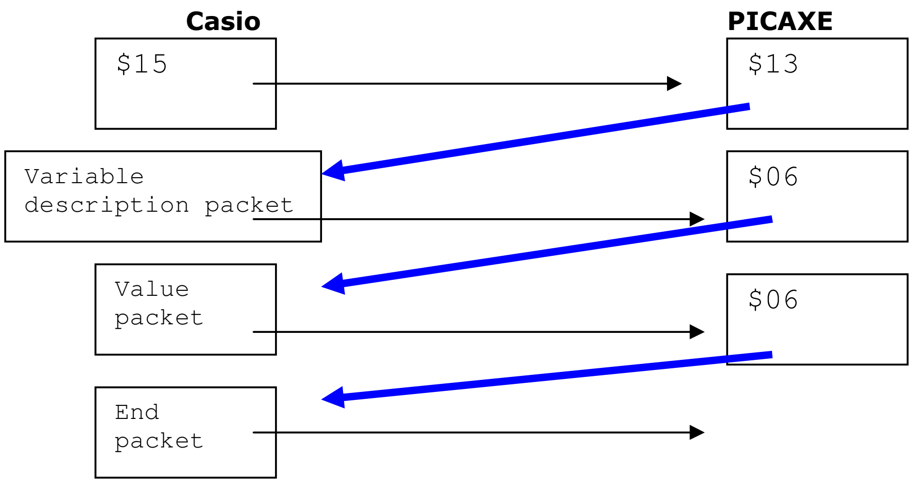
9. Casio 9750G RECEIVE() sequence / protocol

- When the command is issued the Casio sends an "attention request" byte, $15
- The external device must reply with a "device present" byte, $13 within 0.5 ~ 1 second or a **Com ERROR** message is displayed
- The Casio sends a *Request packet* which consist of 50 bytes.
- The external device receives this and confirms with one byte, $06.
- The Casio confirms that it's ready for a *Variable-description packet* with $06
- The external device sends a *Variable-description packet* consisting of 50 bytes. Byte 12 in this packet (variable name) seems to be totally ignored by the Casio, as the variable is stored under the name indicated in the *Request packet* anyway. Still it's recommended to send back the same name/byte as received in the *Request packet* byte 12.
- The Casio confirms that it's ready (for a *Value packet* or an *End packet*, depending on byte 9 and 11 in the previous packet) with $06
- A *Value packet* is then sent from the external device. It is still consisting of either 16 or 26 bytes, depending on whether the variable also contains an imaginary part (complex number) or not. If the variable is empty this packet is not sent at all (length: 0 bytes). This means that the actual "Alpha Memory" variable in the Casio is deleted. This will free 10 (real) or 20 (complex) bytes of memory in the Casio. In calculations the deleted variable's value equals zero (0).
- If a *Value packet* was sent then the Casio confirms (as usual) with $06 (if the variable is empty there is no *Value packet*, and this byte is not sent either.)
- To close the communication cycle the external device sends an *End packet* of 50 bytes. The end packet is always the same; none of the bytes change their value.



10. Casio 9750G SEND() sequence / protocol

- When the command is issued the Casio sends an "attention request" byte, $15
- The external device must reply with a "device present" byte, $13 within 0.5 ~ 1 second or a **Com ERROR** message is displayed
- The Casio starts to send the *Variable-description packet*, which consists of 50 bytes.
- The external device receives this and confirms with $06.
- The Casio sends its *Value packet*, consisting of either 16 or 26 bytes. Packet length depends on whether the variable also contains an imaginary part (complex number) or not. If the variable has never been assigned a value since the last reset of "Alpha Memory" this packet will not be sent at all.
- The external device receives this packet and confirms with $06. (If an empty variable is sent, then there is no *Value packet*, and this byte is not sent either.)
- To close the communication cycle the Casio sends an *End packet* of 50 bytes. The end packet is always the same; none of the bytes change their value.


Casio
PICAXE
$15
$13
Variable description packet
$06
Value packet
$06
End packet