Advanced Programmable Calculator (APCalc)
User Manual

Scientific	Base Conversions	Financial

z=cos(x)+sin(y)

D 1.32892605

sin	cos	tan	asin	acos	atan
log	ln	exp	sqrt	cbrt	n!
Ans	π	y^x	+/-	1/x	Clr	⬅
Sto	7	8	9	*	÷	f
Rcl	4	5	6	+	−	V
QEd	1	2	3	(	)	
R/S	0	.	E	APCalc

45839289492945934

F

D A2 DA97 46FB BC0E

A	B	C	D	E	F
F10	T16		:	CVT	-->
Ans	π	y^x	+/-	1/x	Clr	⬅
Sto	7	8	9	*	÷	f
Rcl	4	5	6	+	−	V
QEd	1	2	3	(	)	
R/S	0	.	E	APCalc

Payments=

D -578.26

Beg	End	--	--	--	--
NP	I	PV	FV	PMT	--
Ans	π	y^x	+/-	1/x	Clr	⬅
Sto	7	8	9	*	÷	f
Rcl	4	5	6	+	−	V
QEd	1	2	3	(	)	
R/S	0	.	E	APCalc

Scientific   **Base Conversions**   **Financial**

Table of Contents

Introduction

Introducing an Advanced Programmable Calculator (APCalc) using Algebraic Entry Notation.   APCalc is now a color or black and white application. If you have a color device, APCalc will load in color. If you have a black and white device, APCalc will load in black and white. The same program works for both screen types.

APCalc, in addition to providing normal calculator functions, allows you create your own custom programs and thereby customize APCalc to your own requirements. You can then save these custom programs in APCalc's built-in program database.

Programs can be grouped into any of nine different User Function Key Groups to allow quickly transforming APCalc from a scientific calculator to a financial, units conversion, currency conversion, base conversion or any other calculator configuration that you desire. Any custom calculator can be restored with two taps of the pen.

APCalc offers features of automatic prompting for new variables, popup selection lists and subroutine calls, branching, looping and comparison operators.

Installation

APCalc consists of two files. APCalc.prc (the application) and APCalcDB.pdb (the program database). There is a third required file, MathLib (a public domain library), that is not part of APCalc. APCalc.prc is used generically to mean either APCalc-C.prc (Color or B&W) or APCalc-BW.prc (B&W only).

These two files can be installed using the Desktop S/W or whatever installation program you currently use, just as you would any other application. These files can be installed in either order.   APCalc.prc and MathLib are the only programs that must be installed.   MathLib must be installed before you launch APCalc.prc, for the first time.

If you do not install APCalcDB.pdb, one will be created automatically when APCalc is executed the first time.

NOTE: Once you have created your own programs, you should not install APCalcDB.pdb that comes with upgrades. If you do, you will overwrite your own saved programs.

- **MathLib** is a **free** shared library that can be used by any OS 2.0+ Pilot program that needs IEEE 754 double precision math functions. It's distributed under the terms of the GNU Library General Public License, and is freely available with full source code and documentation at the MathLib Information web page. It's not a part of the APCalc program, and you're not paying anything for its use; a copy is simply included in this archive for your convenience.

Launching APCalc for the first time

When you launch APCalc for the first time you will see the registration screen, like the one shown at the right. You have two options at this point.

If you have registered APCalc, you should enter the Key that you received with your confirmation, into the Key field, and tap Register. User field and Reg. Code fields cannot be changed. Once the key has been recognized as valid, you will see the "Unregistered", at the top of the screen, change to "Registered" and a popup screen that says "Thank You". Successful registration will remove this screen from being displayed each time APCalc is started.

Alternatively, you can tap Test Drive, without entering a Key, to use APCalc, in its demonstration mode, subject to the limitations of the demo version of APCalc. Demo mode limits the size and number of programs as well as the number of Function Keys. If you have already registered, use Test Drive to exit the screen.

You will then see a screen like the one to the right. This is how APCalc looks with no saved programs. The "Init" is just an initial blank program that is created as the first program name. You may tap Init button followed by QEd to begin editing this program. Programming will be covered later in this document.
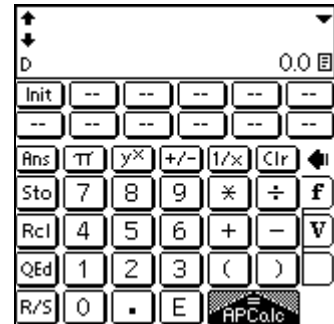
Quick Look

The following information should get you familiar with APCalc. The purpose of this document is to show you how to use APCalc and its functions. It is not aimed at teaching you how to program in general. I will handle, on a case by case basis, any specific questions on how to do specific calculations.
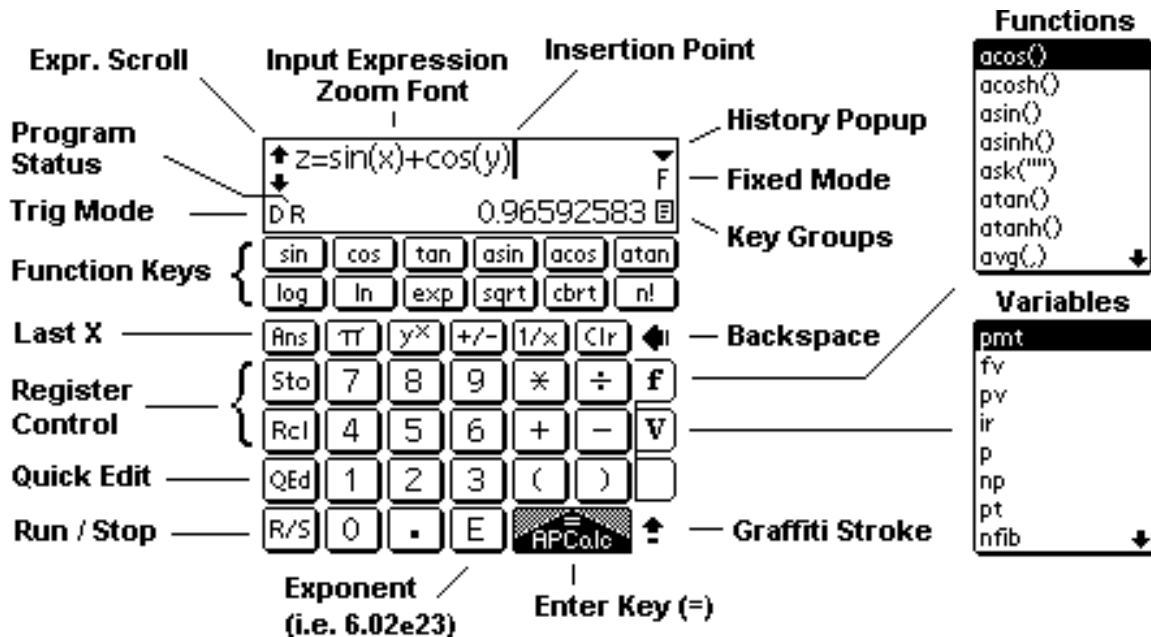
Input Expression – The field in which you enter the expressions you wish to evaluate. Expressions may contain signs, constants, numbers, variables, functions, operators and SPACES (spaces are ignored). You can 'Tap', 'Cut', 'Copy', 'Paste' and 'Select' any part of an expression. Expressions can now be up to 64 characters and may occupy up to three lines. Up/Down scroll buttons (left of input line) have been added to navigate within the expression. Up moves to the beginning of an expression, Down moves to end of expression line.

The input line can contain single expressions or multiple expressions, if separated by a semi-colon. This allows input like: x=5; y=6; z=x*y, on a single line with a result of 30 displayed in the Answer field. See the '$' operator for repeat execution of multiple instruction lines.

Matching parenthesis: As you tap the expression field, you will select data within matching parenthesis. This is useful for deleting parts of expressions within parenthesis.

Status – The current status of the calculator is displayed in this area. '**R**' indicates that a program is running. If a wait() function is executed, the '**P**' (paused) indicates that the calculator is still in run mode and must be continued with R/S. '**S**' indicates single-step mode and requires pressing R/S after each instruction. Single-step mode is set in preferences screen. There is also a Fixed Mode Status on the right side of the screen. It is display a bold '**F**' if the calculator is in a fixed point mode that is other than decimal. Be aware of this status or else you might not be aware that an answer is in some other base.

Trig Mode – '**D**' will displayed for degree mode, '**R**' will be displayed for radians. Tap the '**D**' status field and pop up a selection list (Deg, Rad).

Function Keys (User Defined Keys) – Two row of keys are allocated to built-in functions and User Programs. These function keys can display groups of functions, for easy access. Current function groups include 9 User Function Groups as well as, Trig/Sci, Misc/Hypr, Base 1 and Base 2.

The key label consists of the first 4 characters of the program name. (i.e. Program, will display 'Prog' on the function key associated with the program number).

The user program Function Keys Groups can be renamed using Menu→Edit→Edit Key Groups. Go to the edit screen and choose names you prefer for these groups.

Use the Hard Scroll buttons to change these keys. The state of these function keys is maintained between sessions. You may also change these via the function key popup trigger (to right of Answer) or on the Preference menu. These may be programmatically changed by the function fk(x) where x is between 0 and 13.

Note: Base 2 cannot be accessed via the Function Key Popup; it is an extension of Base 1 and is accessed via Base1, via the hard scroll buttons or with fk() function, in a program.

Last X – The Ans key will evaluate to the last calculated value. Use the 'Ans' variable, in an expression, when you want to refer to the last calculated value.

Register Control – These two keys (Sto and Rcl) are used to store and recall values to and from the 100 built-in APCalc registers. Syntax is Sto(x), where x is any value from 0 to 99. The last calculated value is stored to the specified register.

Quick Edit – This button brings up the currently loaded program, into the editing window. If there is no program currently loaded, tapping QEd will open with a blank program. Use this button to create and edit APCalc programs.

Run / Stop – This button is tapped to begin or halt execution of a previously loaded program. This button serves the same function as menu option, Menu→Program→Run Program (/R). It must be tapped to 'continue' a paused program (i.e. after a wait("label")). A paused program is indicated by a '**P**' in the status area. Tapping R/S can be used to break out of an infinite loop, in a program.

Enter Key (=) – Functions like the $\boxed{=}$ key on any calculator. Tap this key to evaluate the expression, in the expression field.

Graffiti® handwriting recognition – APCalc accepts input from the Graffiti® writing area. Graffiti® characters are indicated in this area.

Functions – Tap this trigger and display a list of functions available for use in expressions. A few functions will be missing from this list because they have no use at the calculator level (outside of a program). Use the popup scroll button to navigate to the function of your choice. Tap that function and it will be inserted into the current expression.

Variables – Tap this button and display the last 30 variables available, for use in expressions. Use the popup scroll button to navigate to the variables of your choice. Tap the variable name and it will be inserted into the current expression. Tapping the right list will allow editing the variable data. When the list reaches 30 variables, new variables are added to the list the oldest will be deleted. Variables are displayed in the order they were created. Variables consist of alpha, numeric and '_' with a maximum length of 20 characters.

Registers – Tap this button to display a list of built-in registers.. Tap the register name and it will be inserted into the current expression. Tapping the right list will allow editing the tapped register.

Backspace – Tapping this key will delete the character to the left of the Insertion Point. Backspace will also set the pending data flag ('pflag').

Expression Result – Result of expressions are displayed in this area (left of Key Groups), whenever the $\boxed{=}$ key ($\boxed{\text{APCalc=}}$) is tapped. Every function will return a value. Expressions like Clear History, Clh(1), will return the last calculated value (Ans). Tapping the Ans field will pop the result into the Expr field.

Note: 'Ans' variable uses the full precision of the previous result, while tapping the Ans field will only use the displayed value (i.e. truncated to displayed precision).

Key Groups – Tap this icon to display a list of Function Key Groups. Select the desired group to assign Function Keys.

Note: Base 2 cannot be accessed via the Function Key Popup; it is an extension of Base1 and is accessed via Base1.

Fixed Mode Status – This status will be '**F**' for any base other than 10. This is to remind you that you are not in the base 10 output format. You can use base(10) as an expression to return to base 10 or you can go to the base function key group and set it there. Since the Base settings are remembered, you can use this status when returning from a previous APCalc session.

History Popup – Tap this button to display the last 32 unique expressions. This can be used for repeat calculation of a recent expression. Tapping any history entry pops that expression to the expression field. The history buffer is maintained between sessions of APCalc.

Insertion Point – The insertion point is a blinking cursor that shows where new text will be entered or the point at which text will be deleted using the Backspace key. You can change the insertion point by simply tapping anywhere within the expression.

Basic APCalc Operation

Basic calculations consist of entering expressions in the expression field, followed by tapping the $\boxed{\text{APCalc=}}$ key. Expressions are entered just as you would write them on paper and may consist of a sequence of signs, numbers, constants, variables, functions, operators and parenthesis. Expressions are evaluated each time the $\boxed{\text{APCalc=}}$ key is pressed. The following are examples of what simple expressions might look like in the expression field of APCalc.

Input Expression	Result
25*56 + 10*1000 `APCalc=`	11,400.00000000
1/2*32*2^2 `APCalc=`	64.00000000
(24 + 23) * 3 + 4 * (15 - 16) `APCalc=`	137.00000000

x = t?; y = 5; z = x * y   OR   prompt for t and display 5*whatever was input. See the
x=t?; x^2; iff(x==0, "end");$ '$' operator for repeat operations.
Note: `APCalc=` implies that the `APCalc=` key is to be tapped rather than writing "APCalc=" into the display.

The number significant decimal digits displayed may vary, depending on the previous preference settings. You may change this selection by activating the preference screen with either the Menu→Info→Preferences or using the shortcut Command (/I) to bring up the preference screen.   Preference selections made are saved and are restored each time APCalc is launched.

Setting Preferences

Deg | Rad	Select degree or radian mode
Fix | Sci |Eng	Select Fixed Point (within 1e8 to 1e-8 limits) or Sci or Eng mode
Decimal	Select decimal display format (0 to 8 decimal places)
SS	Selects single-step mode, during program execution, if checked
TZeros	Trim trailing zeros, of decimal part of result, if checked
RVars	Reset variables each time `APCalc=` is pressed
Display Inst.	Display (animate) instructions, during program execution
Tap Speed	Select fast or slow tap speed for use with Multi-tap selection
Backup DB	Backs up APCalc program database, during HotSync®
ProgHistory	Logs program steps, during program execution, if checked
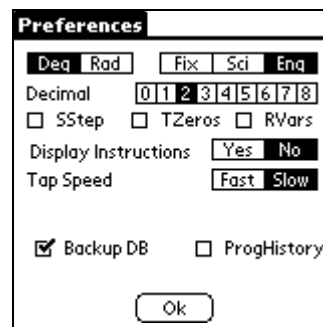
Using variables in an expression

There may be times when you find it necessary to use variables within expressions, at the calculator level (that is, outside of a program).

For instance, to calculate the area of a triangle use the formula ½*base*height. To perform this calculation you would enter:

1/2*base*height (or .5*b*h) and press `APCalc=`.   You will be prompted for base and height, if these variables do not already exist.

Prompting for variables occurs automatically, the first time the variable is encountered. Simply enter the value for the variable and tap Ok. Calculation will continue until complete. The result of the calculation will be displayed in the Ans area next to the Key Group button. **TIP:** using a variable, followed by '?' will force prompting of a variable.
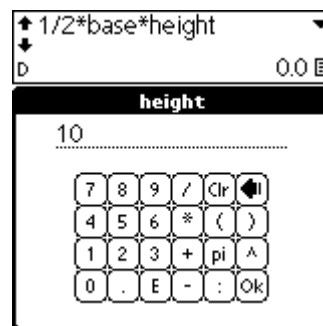
Once the calculation is complete, you can review the variable values by tapping the
[V] tab at the right of the screen. This list of variables will contain the last 30
variables used in all calculations. Tapping on any variable will pop that variable
into the expression field. For instance to calculate the area of a rectangle
(base*height), using the same values and only the variable popup and the
APCalc '*' operator, you could tap: [V] base [*] [V] height [APCalc=]

To delete a specific variable (like base), just enter base= without a value,
followed by the [APCalc=] key. This would delete base and removes it from the
variable popup list. Or, check Del and tap variable name.

You may also assign the value of an expression to another variable. For instance you could have calculated these
two areas and assigned it to new variables, Atri and Arect, by writing the original expression as follows:

Atri = 1/2*base*height   and   Arect = base*height

Another useful way of evaluating expressions is to evaluate a previously entered expression with different variable
values. There are two simple ways to accomplish this. The method you use will depend you your personal
preference.

The first method uses the ? operator. Any variable that has ? appended to the variable name, will automatically
prompt for that variable, even if it already exists. You could have written the equation like:

Atri = 1/2*base?*height? and would have been prompted for base and height each time the [APCalc=]
key is tapped. Or, Arect = base*height? and would have only been prompted for the first occurrence of
base and subsequent values for height.

The second method of forcing prompting for variables is by setting the RVars (reset variables) preference. This will
force the clearing of "all" variables each time the [APCalc=] key is tapped which will force prompting at variables.

Note: Using RVars clears all variables each time [APCalc=] is pressed, use with caution. This applies even if you are
not entering a variable. For instance, if RVars is checked and you simply calculate 2+2, all variables will be cleared.
This is useful, if you have an equation in the expr field and you want to continually be prompted for input. If you do
not use RVars and you are tap [APCalc=] with a previously calculated expression, you will see no change to the
display since it has been calculated with previous variable values.

Using functions in an expression

Using functions is very similar to using variables. Functions may be written into an
expression, selected from the function list, **f** tab, or directly accessed via the
Function Keys. To enter the equation, x=radius*cos(angle), you could
enter:

x=radius*

then select cos() from the popup function list. You will notice that the insertion
point is positioned within the parenthesis. You then write in angle or select it
from the variable popup, if it already exists from a previous calculation. You
would finish the calculation by tapping the [APCalc=] key.
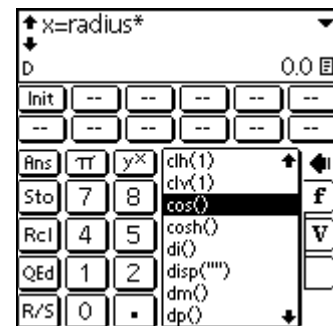
Function Key Groups

Another way of using functions is by directly accessing them via the predefined Function Key Groups. Tap the Key Group icon, to the right of the Answer field, and see a screen similar to the one to the right. Tap Trig/Sci to select scientific functions. This will set the Function Keys to their scientific assignments. Tapping these any of these keys will insert that specific function into the expression field.

This screen shows the default Key Group names. Key Groups UserKeys1 - Subroutine2 can be renamed to anything you require. The last four, Trig/Sci, Misc/Hypr, TVM, and Base1/Base2 are pre-defined, APCalc functions and cannot be changed.
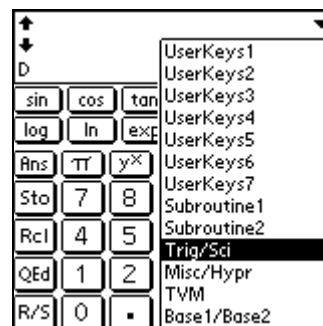
Changing Key Group Names

As you add programs, you will want to change the User Key Group names per your configuration. These names are the names that are displayed in the group list, when you tap the key group icon, on the main calculator screen.

To edit, select Edit→Edit Key Groups from the APCalc menu. You can then replace any default name to something more appropriate to your program groups.

Programs are divided into groups of 12 programs. Group 1 is applies to programs 1 through 12, Group 2 applies to programs 13 through 24, an so on.
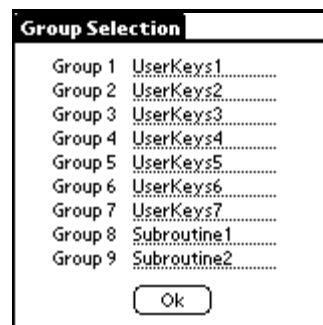
Creating/Editing APCalc Programs

Creating a program is just an extension of the basic operation of APCalc. A program is simply a sequence of basic APCalc calculator expressions that have been saved, as a program. Programs can be created within APCalc, using QEd (quick edit) or they can be written in MemoPad and imported (discussed later) into APCalc. Note: There is a required format to follow, when creating programs within MemoPad. See Import section for this format.

In addition to basic expression evaluation, programs can initialize variables, prompt for user input, jump to other locations within a program, call other programs and display results.

Each program instruction line can contain a single instruction or multiple instructions separated by a semi-colon. A multiple instruction line will execute each instruction but will display only the last result. The LastX (Ans) variable will be set for each individual instruction within a multi-instruction line.

Creation of programs consists of a few simple steps.

Creating or Editing a program involves the same process. The only difference is that creating a new program opens the QEd screen with a blank instruction list.

Tap the QEd button to open the instruction list screen, shown at right, for the new or currently loaded program. An example is shown at right.
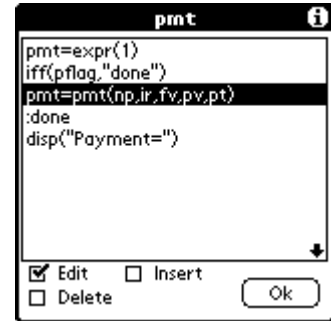
Check Edit, then tap instruction to open edit instruction screen, for that instruction.
Check Delete, then tap an instruction to delete the tapped instruction.
Check Insert, then tap instruction to insert instruction, in front of tapped instruction.
Tap the Ok button to end the editing session and return to the calculator screen.

Once in the Instruction Edit screen, you can use the function list, the keypad or the keyboard to edit instructions. When the current expression is complete you may either tap Done, to return to the Instruction List screen, or you may tap one of the other buttons. Tapping Undo, restores the instruction to its original state.
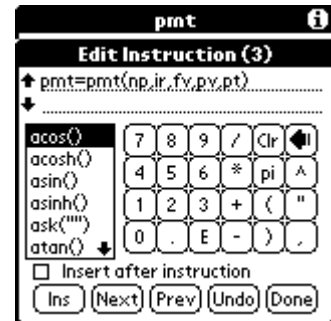
To insert another new instruction, after the current instruction, check "Insert after instruction" before tapping Ins (insert). To insert before the current instruction, leave that box unchecked and tap Ins.

Next, Prev will display the next or previous instruction, if it exists.

Saving APCalc Programs

After you have created your program, don't forget to save the program. At this point, the program is just in program space. If this is a new program, you must use SaveAs. If this is a 'loaded' program (via Load or after executing a program), you can use Save to change, or SaveAs to save under a different name. If you don't perform a Save or Save As, the program will be lost when you exit APCalc or execute a different program.

All of the first 108 user generated programs are automatically assigned to the User1 through User 9, upon initial creation.

Changing the order of functions on keys requires moving the program location. See the "Import/Export to and from MemoPad" section for instructions in moving programs.

Example Programs

Example 1: Compute the area and circumference of a circle

Here is an example of a program that computes the area (A=pi*radius^2) and circumference (2*pi*radius) of a circle and prompts for circle radius. It will also pause and display a label before proceeding with the circumference calculation.

```
radius=
pi*radius^2
wait("Area =")
2*pi*radius
disp("Circumference =")
```

Example 2: Convert between Fahrenheit and Centigrade

Example of a program to convert between Fahrenheit and Centigrade. This example uses the iff(expr, "label") function. Comments (//) are not entered into program. This program will display a popup that offers a choice of F to C or C to F. Select 1 to convert from F to C or 2 to convert from C to F.

```
:Temp Convert
F=                      // clears F; forces prompt of F
C=                      // clears C; forces prompt of C
in("1) FtoC  2) CtoF")  // prompts for choice
sto(1)                  // saves choice
iff(rcl(1)>1,"CtoF")    // tests for choice = 2
(F-32)*5/9              // executed if choice was 1
wait("C = ")            // displays C = label
jmp("done")             // Jump to done
:CtoF
9/5*C+32                // calculates F
wait("F = ")            // displays F = label
:done                   // done (comment)
```

Example 3: Single-line programs

The User Keys have two modes. For multi-line programs, the program is **loaded and executed** as soon as the User Key is tapped. For single-line programs, the **= key must be pressed** before the expression is executed. This second mode is to allow you to store constants, or single line expressions for inclusion in other expressions. Any single line expression can be executed immediately simply by adding a comment line as the first line, thereby, making it a multi-line program.

To assign a User Key to perform a sin( ) function exactly like the sin( ) on the Trig/Sci group, you would create a single line program like:

```
sin()
```

as the only instruction. Assign it a name of, say, sin. 'sin' would then be assigned the next available User Key. When that key is tapped, sin() will be inserted at the insertion point in the expression. APCalc= would be required to evaluate the expression.

Note: Because of limited number of programs and Function Keys, single-line behavior is not available in unregistered versions. Single-line programs behave just like multiple line programs, in unregistered versions.

Example 4: Multiple-line programs

To assign a User Key to perform a clv(1) function, and execute immediately by tapping the User Key, you could create a multi-line program like:

```
:begin
clv(1)
```

as the program. Assign it a name of, say, clv. 'clv' would then be assigned the next available User Key. When that key is tapped, clv program would be executed immediately.

Example 5: Subroutine / Jump Examples

The four programs below illustrate the use of call("prog") function as well as one use of the ask("prompt") function. Also you will see an example of out("prompt", expr). Program 1 (PG1) calls program 2 (Sub2) to initialize two variables, then the program returns to program 1 to calculate x+y. Following that calculation, program calls program 3 (Sub3) to calculate x*y. Program 3 then calls program 4 to calculate x*y/x+y. Finally the programs return back through 3 to program 1. ##Names are not part of the programs, while in APCalc.

```
##PG1                                      ##Sub2
:PG1                                       :Sub2
45*25                                      x=45
call("Sub2")                               y=25
x+y                                        disp("Sub2 (init): x=45, y=25")
out("PG1: x+y=", x+y)                       ask("Continue")
ask("Continue")
call("Sub3")
x*100+y
disp("PG1 End: x*100+y=")

##Sub3                                     ##Sub4
:Sub3                                      :Sub4
out("Sub3: x*y=", x*y)                     out("Sub4: (x*y)/(x+y)=",(x*y)/(x+y))
ask("Tap any key")                         a=Ans
a=Ans                                      ask("Exit")
call("Sub4")                               a
a
```

It is important to note that call() does not return a value that is the result of the called program. What call() returns is the last calculated value at the time of the call. So, for the PG1 example above, if 3rd instruction was result=call("Sub2"), the value of result would be 1125 (the result of 45*25). If an instruction like, result=Ans were inserted right after the call(), the value of result would be zero, since that is the last calculated value, as a result of Sub2.

Call Variation
There is a variation of the call("program") that can be used in conjunction with the ask("") function. This function is callif(index, "prog0", "prog1", "prog2", "prog3", "prog4"). This variation uses an initial parameter index to select which program will be called. The index is zero based. If index is 2, then "prog2" will be called, and so forth. If callif() is called with a missing index parameter, it will execute exactly as call() but will use "prog0" name. Any program names may be used for program parameters.

Example (pick list to select running Sub 2 or Sub 3 above):

```
:callif test
a=ask("Run Sub2", "Run Sub3")
callif(a, "Sub2", "Sub3")
```

Jump Variation
There is also a similar variation to the jmp("program") function. The function jmpif(index, "label0", "label1", "label2", "label3", "label4") greatly simplifies branching using the ask() function or other ways that depend on an index.

Executing APCalc Programs

All programs that are loaded into APCalc are assigned to one of the Function Keys. Simply tap that key to execute the program. When a program is executed, it is also loaded into the program space and can be edited via the QEd button. Once a program is loaded into the program space, it can also be executed, using R/S .

That's about all there is to creating and running a program.

APCalc Editing Tips

Graffiti® command stroke:
APCalc uses Graffiti® command strokes just like any other application. There are, however, a few strokes that are very useful to APCalc. These include:

Move cursor one char right	right-left-right
Move cursor one char left	left-right-right
Delete last entry (char)	left stroke
APCalc= key	upper-right to lower-left
Select All	/S, selects the entire expression
Copy (with nothing selected)	Copies Ans to the clipboard.

Tap Selecting Text:
I have added the ability to select within matching parenthesis. This is a multi-tap selection.

Tap once:	Sets the insertion point
Tap twice:	Selects a word or number
Tap again:	Selects all text within matching parenthesis
And so on….	

If there are no matching parenthesis, all remaining text is selected.

This feature works on expression field on calculator, Full Screen Program Edit mode, and Instruction Edit Mode (QEd).

APCalc Programming Tips

1) One of the first things I do is create dummy programs for all 108 locations. The dummy programs are nothing more than a comment. That way, I can just select the program I want to edit and avoid having to move programs around.

   :dummy
   disp("not programmed")

2) Using var=, at the beginning of a program, will clear the variable and force its prompting, if that variable appears later in the program. If it used at the end of a program, it will remove the variable from the variable popup list. You can use this to remove internal variables from the popup list.

3) Using var?, prompts immediately for a variable input. This does not delete the variable first. The result is that if the variable already exists, its place in the variable popup list, will not change. If it does not exist, one will be created and placed at the top of the list. Use this to control the order of prompting of variables in a program or equation. var? can also be used within an expression (i.e. z =x?+y?)

4) The purpose of expr(1) is to grab a pending expression from the input field. There are two cases where you may wish to do this. You can use this to have a program be dual function (store a value, calculate a result).

    a) Case1 (to grab input at beginning of program): The expr(1) function should be the first instruction. If it is not the first instruction, the Prefs→Display Instructions set to YES, can change the input and you will get an undesired result.

    b) Case2 (to grab input in the middle of a program): The expr(1) instruction must be preceded with a wait("prompt").

Note: If you intend to grab pending data, you must include this function as the first instruction.   You should also always use iff(pflag, "label") to test for the existence of pending data.

5) pend(1) is similar to expr(1) except that it grabs data that was pending, at the time the program was executed. pend(1) does not have to be the first expression. Wherever it is located, will return the pending expression, at the time the program began.

6) Use something like x=Ans, as the first instruction in a program, to have one program continue with (and save) a calculation from a different program. You can have programs who's input is dependent on the result (output) of another program.

7) Other examples can be found at http://www.halcyon.com/ipscone/apcalc/examples/. Navigate to Documentation→Program→Examples.

Base Conversions

APCalc offers flexibility in the way you can handle base conversions and fixed point math. There are two distinct ways to handle base conversions and fixed point math and operations.

You can perform all fixed point operations and conversions using the Base Conversion Function Keys. This mode provides you with the greatest precision. You have the use of the full 64 bit fixed point number system.

You may also mix fixed point calculations and floating point operations using a special notation discussed below. Using this mode, you are limited to fixed point numbers less than 1e53-1. All fixed point numbers are unsigned 64 bit numbers.

Number bases from 2 to 36 can be handled by APCalc. Numbers larger than base 16 use alphabetic letters g-z. All base numbers are displayed using this character set. 0123456789ABCDEFghijklmnopqrstuvwxyz

"**F**" just below the history popup trigger, indicates that you are still in some Fixed Point Base other than 10.

Base Conversion Key Group Definitions

Base 1

A-F   Hex characters used to input Hex numbers, for numbers 10 to 15.

F10   **FromBase** Key: Sets the input mode. Enter a number between 2 and 36 and tap the F10 key (FromBase) key. It will rename itself to denote the current From Base setting.

T10   **ToBase** key: Sets the output (Ans) mode. Enter a number between 2 and 36 and tap the F16 (ToBase) key. It will rename itself to denote the To current base Setting.

F<>T   This button does a quick swap of FromBase and ToBase.

:   This is a special operator, used in floating point mode, to provide limited base conversion capability in floating point mode. Use it to enter numbers in a particular base like: 10:255, 16:3FF, 2:10110, etc. In floating point mode, that is when you use APCalc= to evaluate an expression, all expressions contain functions, constants and decimal input. You cannot input numbers in Hex, for instance, without using this operator.

CVT   Converts a number, in proper format, according to FromBase setting to a result displayed in the ToBase format. Enter a number in the proper format and tap CVT to convert it.

➔   Switch to other base Function Key Group.
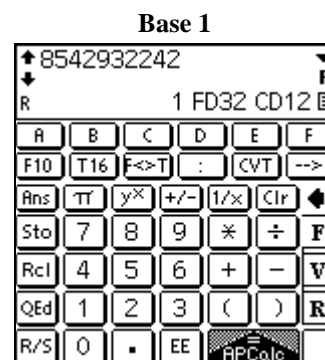
B+   Adds 'expr' to the last displayed value. 'expr' is in the FromBase format selected with the FromBase key.

B-   Subtracts 'expr' to the last displayed value. 'expr' is in the FromBase format selected with the FromBase key.

B*   Multiplies 'expr' to the last displayed value. 'expr' is in the FromBase format selected with the FromBase key.

B/   Divides the last displayed value by 'expr'. 'expr' is in the FromBase format selected with the FromBase key.

<<   Shifts the last displayed value, left, by n bits. Enter 'n' then tap the << key to shift left n bits.

>>   Shifts the last displayed value, right, by n bits. Enter 'n' then tap the >> key to shift right n bits.

and   Performs a bitwise 'and' operation of the last input value and the 'expr' displayed in the input field.

or   Performs a bitwise 'or' operation of the last input value and the 'expr' displayed in the input field.

xor   Performs a bitwise 'xor' operation of the last input value and the 'expr' displayed in the input field.

not   Negates the last displayed value. Does not require an input value. Just tap to negate last Ans.
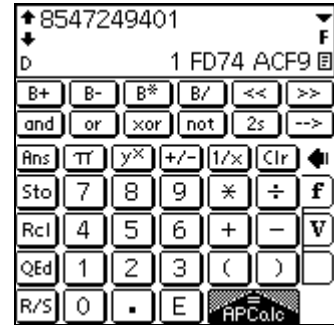
Fixed Point Calculations/Conversions

All conversions and fixed point math are performed using a FromBase value and a ToBase value. Before a conversion can take place, you should enter the correct FromBase and ToBase, using the Fxx (temporary name) and the Txx (temporary name) keys. The FromBase (Fxx) and ToBase (Txx) are remembered so F10 and T10 will be remembered between APCalc sessions. Check the 'F' (fixed point status) if you need to know whether or not you have previously set a Fixed Point base.

Example 6: To convert 3FF from HEX to DEC, you would enter the following:

```
1) 16        // base 16
2) Tap F10   // key is renamed to F16 (FromBase)
3) 10        // base 10
4) Tap T10   // key is renamed to T10 (ToBase)
5) 3FF
6) Tap CVT   // Convert - see results (1023) displayed
```

To continue and convert 3DACE from HEX to DEC, just:

```
1) 3DACE
2) Tap CVT   // no need to reenter FromBase and ToBase values.
```

To then see what this value might be in BINARY, just change the ToBase value:

```
1) 2
2) Tap ToBase key // ToBase key is renamed to T2
```

To return to base 10 output:

```
1) 10
2) Tap ToBase key // to return to DEC output display
```

To then see what this value might be when 3FF added to it:

```
1) 3FF
2) Tap the --> key       // switch button groups
3) Tap the B+ key // result of expr + previous value
```

NOTE: B+, B-, and, <<, etc expect the input data in the FromBase format. (i.e. FF would not be an acceptable input if the FromBase was F10).

To then shift the results left by two bits:

```
1) 2  // shift left 2 bits
2) Tap the <<      // results displayed in Ans field
```

'and', 'or', 'xor' operate in a similar manor. The 'not' functions does not require an input. Just tap the 'not' key to calculate the inverse of the displayed value.

Note: All of these operations expect the input expression in the correct format as determined by the FromBase and ToBase input, using the FromBase and ToBase keys.

Mixed Fixed Point and Floating Point operations using the ':' operator and the Equal key:

This mode allows limited use fixed point numbers, in various bases, to be intermixed in the basic floating point calculator mode. **In this mode ONLY the ToBase format is used.** You can use this mode to calculate a mix of base values and display the results in another base. The syntax for this operator is:

```
base:number
```
where base is a number between 2 and 36 and number is the number in that base, in the proper format.

Example 7: Using base operator ':'

```
1) 10:255           // 255 as a base 10 number (DEC)
2) 16:3FF           // HEX number 3FF
3) 16:255           // HEX number 255
4) 2:101010         // BINARY number 101010
5) 27:g3F2          // g3F2 in base 27 (whatever that is good for)
```

You can mix and match these numbers and evaluate expressions using this notation.

Example 8: Mixing bases; 118(DEC) + 3FF (HEX) + 101010 (BINARY) and display as a HEX

```
1) 16
2) Tap ToBase key        // F16 in legend, sets output to HEX
3) 10:118+16:3FF+2:101010
4) APCalc=
```

This may seem awkward at first but I think you will find this very useful and provides a great deal of flexibility when dealing with math operations on mixed, fixed point numbers.

Note: If you type FE3 (what you think is a HEX number) and tap APCalc= , APCalc will think this is a variable. You must use the ':' operator if you wish to use the basic calculator with fixed point numbers.

This mode is limited to Fixed Point Numbers < 1e53-1. That is the highest fixed point number that can be accurately represented by the floating point format used by the OS.

Financial Programs

The example demonstrates the use of the built-in Time Value of Money (TVM) functions

Financial Key Group Definition

Beg   tap if payment is at the beginning of the month.
End   tap if payment is at the end of the month
DR   Discount Rate (used with Net Present Value functions)
CF   Cash Flow input (used with Net Present Value functions)
NPV   Net Present Value (calculates NPV given CF inputs)
IRR   Internal Rate of Return (calculates IRR given CF inputs)
NP   enter or calculate the total number of payments to be made
IR   enter or calculate the periodic interest rate (i.e. 7.5 is entered as 7.5/12 if the period is monthly)
PV   enter or calculate the present value of a loan
FV   enter or calculate the future value of a loan
PMT   enter or calculate the periodic payment of a loan. Payments are negative while cash received is positive.
CLF   Clears financial parameters.
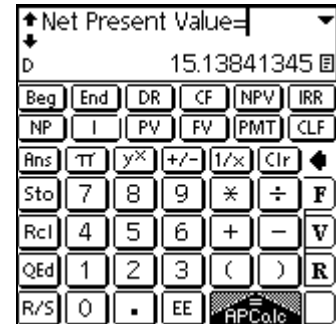
To use these programs, make sure RVARS is UNCHECKED.

Assume a 150,000 loan at 6% for 30 years and you wish to know the monthly payments, with payments made at the beginning of the month. What is the monthly payment?

Number of payments	360	enter 360 and tap 'NP'
Yearly Interest	6.5%	enter 6.5/12 and tap 'IR'
Present Value	150,000	enter 150000 and tap 'PV'
Future Value	0	enter 0 and tap 'FV'
Pmt at beginning of mo.		tap 'Beg'
Payment	?	tap 'PMT' to calculate –942.99

The result will show payments of –942.99. Remember payments to you are positive and payments from you are negative. So, when entering 'PV' and 'FV' the values are positive and your monthly payment is negative.

To see what the payment would be, if the interest was 6% instead of 6.5%, just enter 6/12 and tap 'IR' followed by tapping 'PMT' and see that the new payment is, -894.85.

If the payment was at the end of the month, just tap 'End' followed by 'PMT' and see, -899.33.

Example 9: TVM programs (NP, IR, PV, FV, PMT, BEG, END)

While the TVM programs are built into APCalc, the following examples show how APCalc can be used to create specialized programs to perform just about any task you have. The example below, was developed to fit on one Key Group but you can assign the programs to any keys you like. These example programs require APCalc version 1.98f or above.

The following programs can be entered into APCalc to provide baseline TVM calculations that can be modified to add additional financial enhancements.

```
##Beg        // name not part of program
:beg
pt=1
```

```
##End        // name not part of program
:end
pt=0

##NP         // name not part of program
np=expr(1)
iff(pflag,"done")
np=np(ir,pmt,fv,pv,pt)
:done
disp("Number of pmts=")


##IR         // name not part of program
p=expr(1)
iff(pflag,"in")
disp("wait...")
ir=ir(np,ir,pv,fv,pmt,pt)
jmp("done")
:in
ir=p
:done
disp("Interest Rate (per period)")


##PV         // name not part of program
pv=expr(1)
iff(pflag,"done")
pv=pv(np,ir,pmt,fv,pt)
:done
disp("PV=")


##FV         // name not part of program
fv=expr(1)
iff(pflag,"done")
fv=fv(np,ir,pmt,pv,pt)
:done
disp("FV=")

##PMT        // name not part of program
pmt=expr(1)
iff(pflag,"done")
pmt=pmt(np,ir,fv,pv,pt)
:done
disp("Payment=")
```

NPV Calculations

Both NPV and IRR calculations require inputting Cash Flow data. This cash flow data is loaded into APCalc registers, beginning with Register 1 and continuing until all cash flows are entered, up to 98 values. Begin all NPV and IRR calculations **by clearing** the financial registers (CLF). Note: this does not clear the registers but only the financial parameters, pv, ir, ect.

Register locations:
Register 1:   Initial investment (usually a negative number, since cash if flowing away from you).
Register 2:   First cash flow input
Register 3:   Second cash flow input
Register N: Nth cash flow input
Register N+1:   Net Present Value (also displayed in the APCalc Ans field.

To look at an Investments. Using a 10% cost of capital (discount rate, DR) , the present values of inflows you would begin this calculation by clearing the financial registers with CLF. Then input the discount rate of 10% by entering 10 followed by tapping the DR key. Then input each of these inflows below, followed by the CF key.

Investment

Year	End of year cash flow	NPV at end of year
2000	-1,000,000	-1,000,000
2001	400,000	-636,363.65
2002	400,000	-305,785.12
2003	400,000	-5,259.2
2004	400,000	267,946.18
2005	400,000	516,314.71

Now that you have entered the last value for 2005, tap the NPV key to compute the Net Present Value. You want to see a positive for good investments. Larger, in general, is better than smaller NPV.

NPV   516,315

IRR Calculations

To determine the Internal Rate of Return, simply tap the IRR key. This calcualtion uses an iteration process of 20 loops and produces an approximate result that should be good enough for any use but does not return the precise exact IRR. It would be too slow, if I processed it to conclusion and would add no value, in doing so.

IRR can be in the range of -5010 to 5010. If you get a return value that is at these limits, then your answer is larger or smaller than the returned value.

Statistical Functions

The statistical key group provides functions including standard deviations and linear regression as well as correlation coefficients. This section will show how to use the key group to provide this capability. APCalc statistical group can handle 1 and 2 variable statistics.

Keys:
Init   Sets 1 or 2 variable statistics. Select 1 for 1 variable statistics and 2 for 2
         variable statistics. You may or may not want to clear data.
x,y+   Adds each data point to the set OR displays sum of x values.
x,y-   Removes data point(s) from set OR displays sum of y values.
n!   Computes n factorial
Regs   Display the statistical registers
n   Displays the total number of data points
avg   Displays the avg of x and y
SD-S   Computes the standard deviation of x and y, using the Sample method
SD-P   Computes the standard deviation of x and y, using the Population method
L.R.   Computes the linear regression of a set of data. Computes the slope and intercept of the best matching line.
y,r   Computes the estimate of y given x as an input. Also, displays the correlation coefficient for the point.

Example 1: Compute the standard deviation of a given set of numbers using single variable statistics.
Begin by initializing the statistic functions. For 1 variable statistics, enter 1 and tap Init.

Data set: x= 5.5, 6.9, 10.3, 15.6, 21.2

Enter each of these values followed by tapping the x,y+ key to enter each piece of data. When all of the data is entered, you can compute:

SumX = 59.5 by tapping x,y+
avg(x) = 11.9 by tapping the avg button
SD-S = 6.49422821 by tapping the SD-S button
SD-P = 5.80861429 by tapping the SD-P button

Example 2:
Given data set:
x = 5.5, 6.9, 10.3, 15.6, 21.2
y = 7.8, 8.2, 10.2, 18.7, 20.3

Begin by initializing the statistical group to handle 2 variable statistics. Enter each x followed by tapping x,y+. The y value is entered at the popup. When finished entering all data you can compute:

SumX = 59.5 by tapping x,y+
SymY = 65.2 by tapping x,y-
avg(x) = 11.9 by tapping avg, avg(y) = 13.04 by tapping the popup pause indicator
SD-S for x = 6.49422821 by tapping SD-S, and SD-S for y = 5.99357990 by tapping the S.D.(y) popup
SD-P for x = 5.80861429 by tapping SD-P, and SD-P for y = 5.36082083 by tapping the S.D.(y) popup
Slope of closest matching line = .89353883 by tapping the L.R. button,
Intercept (B) = 2.40688797 by tapping the popup

To estimate y given an x, enter the x value followed by y,r. Tap the popup to get the correlation coefficient.

To display all of the statistical registers, tap the Regs key.

Matrix Operations

These functions will allow manipulation of up to 3 matrices (0 to 3). The default value for all 3 matrices is 3x3. Matrices can be sized, for now, to 5x5 size. The matrices use the existing storage registers. Matrix parameters range are:

# of matrices	- 3
number of rows	- 0 to 5
number of cols	- 0 to 5
row index	- 0 to 4
col index	- 0 to 4

These functions include:
InitMatrix   mtxi(matrix, row, col, start)

Use this function to size a matrix. Matirces are not checked for overlap, as yet, so that will be up to the user to make sure that matrices do not overlap.

Ex: mati(0, 4, 5, 20)   defines matrix 0 as 4x5 and starts at reg 20
          mati(1, 2, 2, 50)   defines matrix 1 as 2x2 and starts at reg 50

Return value is the dimension + start register in the following format:
r.cSR where r is row, c is col and SR is starting register.

GetMatrixValue   matg(matrix, row, col)

Use this function to get a matrix value.
Ex: mtxg(2, 3, 4)   returns the value of matrix 2 at row=2, col=3

SetMatrixValue   mats(matrix, row, col, value)

FillMatrix   matf(matrix, value)
Use this function to set a matrix value.
Ex: mtxs(2, 3, 4, number) the value of matrix 2 row=2, col=3 to number

DispSizeMatrix   matz(matrix)

Mostly used for debug. This function returns the size of the matrix as specified in mtxi but sets nothing. Reg 97=row, 98=col, 99=start.

Graphing

Graphing on APCalc might be different that you have seen on other calculators. APCalc graphing has been designed to allow graphing of equations, without the need of programming experience. You may, though, use programs to create more complex graphs and to automate many of the graphic setup steps.

Manual Graphing of an equation

It is quite simple to graph an equation.   Just perform the following steps and you will begin to graph equations, without any programming experience.

1)   Enter graph(1) and tap APCalc=
2)   Select Setup from the menu bar (/S)
3)   Write in the equation on the f(x) line (i.e. sin(2*x))
4)   Set: xMin, xMax, yMin, yMax, step
        xMin = 0
        xMax = 360
        yMin = -1
        yMax = 1
        step = 8
5)   If you wish to display x and y axis lines, check the appropriate checkbox
        x Axis on = checked, y Axis on = checked, "Report Errors" is used to
        suppress errors when discontinuities are encountered.
6)   Select "Exit" button to return to the graphic screen
7)   Tap "Plot" to plot the equation (see image at right)
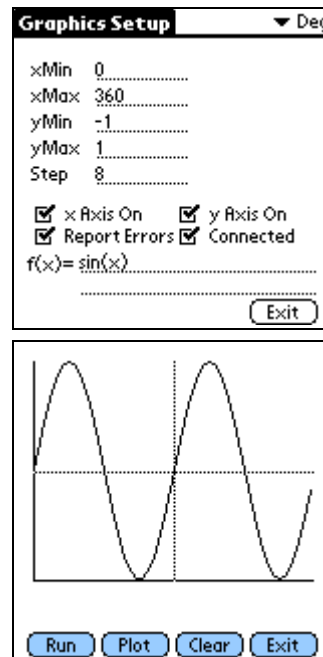
That's it! Very simple. Note, however, if you select an inappropriate xMin, xMax, yMin, yMax, or step, you might get undesired results or the plot may not be visible at all.

If you graph a function, and the function uses x values that produce an undefined result, you will get an error popup, as a warning (i.e. tan(x) where x = 90 deg). Also note that if the x values step from 0 to 180, depending on the step, you might not exactly hit on zero (for the tan example). You just need to be aware of this.

Graphing using a program

Graphing using programs is just like any other program, except that graphing programs use a special set of instructions (see graphing instruction list at end of document) and programs are run on a different screen.

Using a graphing program, you may perform all of the steps that you can do manually. There are a few additional instructions that are for color, and not available in manual graphing. This functionality will be available in a later version. Also note that some of the functions operate differently on black and white than they do on a color device. For instance the rect(3, 3, 10, 10, 1) will fill a rectangle with the foreground color. This may look fine on a color device but may look differently on a black and white device.

Graphing program example: Create the following program and assign it to a function key. When completed, tap the function key to execute the program. You should see the graph shown on the image to the right of the program. Note that all graphing programs begin the graphing part with the graph(1) command.

```
:text draw                              (continued from previous column)
mode=dm(-1)
graph(1)                                :dm(1)
gcolor(0,0,255,0,0,0)                    dm(mode)
grect(1,6,8,154,43,0)                    gcolor(0,0,0,0,0,0)
gcolor(255,0,0,255,255,208)              grect(1,6,46,154,81,0)
grect(0,6,8,154,43,1)                    gcolor(0,255,0,0,0,0)
gprt(10,8,"Value1 =")                    grect(1,6,46,154,81,0)
gprt(112,8,"(125-150)")                  gaxis(0,720,-2,2,20)
val1=125                                 gwin(5,45,149,37)
gcolor(255,0,0,255,255,208)              gcolor(0,0,0,0,255,0)
gprt(48,8,val1,4)                        gplot("sin(x)","x",0)
val2=val1                                gwin(5,45,149,37)
:loop                                    gcolor(192,192,192,0,0,0)
gcolor(255,255,255,0,0,255)              grect(1,6,84,154,119,1)
val2=val2+2                              gcolor(0,0,204,192,192,192)
gprt(10,19,"Value2 =")                   grect(1,6,84,154,119,0)
gprt(48,19,val2,4)                       guln(3)
gcolor(0,255,0,0,0,0)                    gprt(66,96,"APCalc")
gprt(10,30,"V1+V2 =")                    gcolor(0,0,0,255,255,255)
gprt(48,30,val1+val2,4)                  guln(0)
iff(val2<150,"loop")                     gwin(10,10,140,110)
```
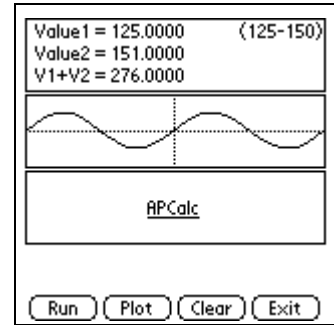

Multiple Value Output Ex.
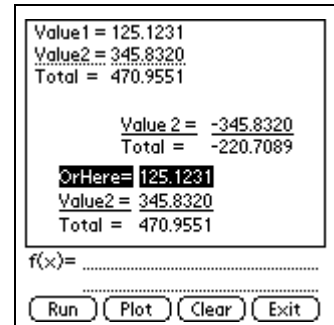
The example above demonstrates the use of graphing in a program. It is important to note that you may use the graphing screen to display formatted text.   The lower image, shown above, demonstrates the use of the graphing screen to display multiple output values.

Note: When you have a program that graphs multiple equations, only the last equation is saved to the f(x) function that shows up on the setup screen.

Import/Export Functions

Beginning with version 2.0, APCalc includes Import/Export functionality. This function also allows "Loading" of a program into program space, "Deleting" of a program or programs and "Import/Export" of programs.

Loading - Loads a program into program space for editing or executing.
Deleting - Deletes individual or groups of programs.
Import/Export - Transfers programs to and from the MemoPad program database.

The Import/Export screen is accessed via the Menu in APCalc. Select Menu→Import to bring up the import/export screen. When the import/export screen is displayed you will see two lists. The list on the left contains all the program groups in APCalc database. As you tap any of the groups, you will see the individual programs displayed on the two rows of function keys at the top of the screen. The list on the right is titled MemoDB and displays up to 108 program groups in the selected category.

Load

If you wish to load a program for editing, simply go to the Load/Import screen and check "Load" followed by the Group and then the program on a particular button. You will be returned to the main screen where you can execute the program with the R/S button or edit it with the QEd button.

Import/Export

Import/Export provides backup of APCalc programs to MemoPad. Programs can be transferred to/from and edited on PC desktop computer, using any software that will allow creation of MemoPad files. The format of the group files, in MemoPad, must conform to a specific file format, which is shown below. These programs can be imported into APCalc. This capability also allows programs to be directly created within MemoPad and imported into APCalc. There are some limitations that you should be aware of when using MemoPad (see limitations at end of section).

Importing is accomplished by importing entire groups. Groups may contain from 1 to 12 programs

The programs, within each group file, overwrite (or overlay) the corresponding program that is already loaded on APCalc. It is not necessary for a complete set of 12 programs to be contained within a group. If your program group has only, say, 3 programs, ONLY those corresponding programs in the selected group will be overwritten. The other 9 programs will remain unchanged.

You may use multiple group names to load a particular group. For instance you might have a group called StatsA, which contains programs for the first 6 programs for group 1. You might also have another group called StatsB, which contains programs for the second 6 programs for group 1. You can then import StatsA, which will fill programs 1 to 6 and StatsB, which will fill programs 7 to 12.

Note1: When the import screen opens, it will open with the current key group selected. This makes it easy to install a particular program group. Just select the Key Group to which a program will be added and when you open Import that key group is displayed in the left list.

Note2: If you have MemoPad records marked as _private_, they will not show up in the import side of the screen.

To Export a single program group, select the Category, tap the group and button to export, then tap →. **Note: You may tap any of the buttons shown on the 2 rows of function keys. Any key not highlighted, will not be exported as part of the key group.** A new program group is appended to the current category. Since a single program may contain less than the total of 12 programs, you will be prompted for the export group name. This will allow you to save partial groups under different names than full groups.

To Export All, select Category, tap the BOLD → button, to append all APCalc groups to current category. Use Export All, to back up your current configuration.
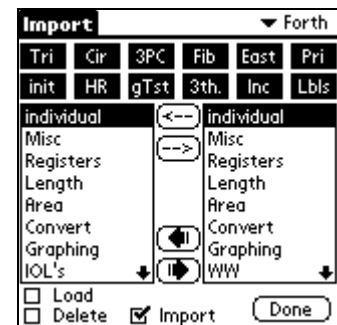
To Import a single program group, check Import checkbox, select category, tap the MemoDB group file to import, tap the group destination, then tap the ← button to import a single program group. Any program within the imported group, will overwrite the corresponding program on the destination group.

To Import up to 9 program groups within a category, select the category, tap the start group destination (right list), then tap the destination group, then tap the BOLD ← button, to import up to 9 groups. The groups will be imported until the last group is imported. You may start at group 1 or any other group but the importing will stop as soon as group 8 is filled. This capability is to allow single button backup/restore operation.

To Delete a group from the APCalc database, check the Delete checkbox, select each program (button), then tap the group name in the left program box. You may delete the entire group or individual programs using the button selections. If all programs are deleted, the Group name will be changed to --- Blank ---.

Delete

To Delete a group from MemoPad database, check the Delete checkbox, then tap the MemoDB list on the right. To delete an individual program in the MemoPad group files, it is necessary to edit that record in MemoPad.

To Delete All APCalc or MemoPad programs use the Menu→Clear All Programs or Menu→Clear All Import

Import/Export allows the use of MemoPad categories so that backup can be performed to any category that you like. You can even edit, add, delete and rename any of the categories and they will be saved to the MemoPad database.

Any number of groups may be saved to each category, **But only 108 are displayed in a particular category**. You can backup to a single (or multiple) categories. Buttons are numbered from 1 to 6 on top row and 7-12 on 2nd row.

Note: This capability (import/export) is only available to registered users.

Import/Export Limitations, Using MemoPad

MemoPad has a 4k record limit that prevents editing of APCalc programs, once the program grows beyond 4k. This presents two problems in using MemoPad, for Importing and Exporting of APCalc programs (Editing and Backup). Fortunately, most user programs will fit within 4k.

EDITING:
There are two ways of dealing with this. The first is to just use MemoPad for Import and Export, without editing within MemoPad. This will still allow you to browse the programs up to about 32k. I think this may only be possible with the new Operating Systems, however.

If you have a large program group that you would like to create or edit from within MemoPad, you will have to break it up into sub groups. For example, say you have a group that will ultimately be about 6k. You can have GroupA (buttons 1 to 6) and GroupB (buttons 7 to 12).

BACKUP:
You can still backup APCalc programs. APCalc has it's own database, for currently loaded programs. This will be backed up, if you have the backup preference bit set.

Any files that are Exported to MemoPad will be backed up, assuming you have chosen to backup MemoPad files, up to 32k. These files will be transferred to the PC and can be restored using the HotSync® manager. However, if you edit or try to edit these files in the PC Desktop, they will be truncated to 4k.

Editing APCalc programs in MemoPad

Programs may be edited on a PC, using any software that will allow files to be saved to MemoPad, and transferred to the device using Desktop. There is a required format that you must follow to allow programs to be transferred from MemoPad to APCalc. There is no format required for programs edited within APCalc.

The first line of a program, edited in MemoPad or on the PC, must begin with two number symbols (#) then the group name, like:

##Key Group Name (up to 16 characters may be used), ex: ##Statistics

The next line is (optionally) blank. The 3rd line is of the format:

##Function Key Label (4 characters only will be used), ex: ##Sdx

The next line contains the Function Key number for the program, with the following format:

##Key Number (00 to 12), ex: ##02

An example of a group file might be like the following:

##Length

##m
##01
instruction 1
instruction 2

##cm
##02
instruction 1
instruction 2

##mm
##03
instruction 1
instruction 2

.
.
.

##nm
##12
instruction 1
instruction 2

A group can contain from 1 to 12 programs. You do not have to use key numbers 1 to 12 in sequence. For instance, in the example above, you might have only 4 programs. Program key numbers might be 03, 05, 07, 12. In that case the four programs would be loaded to function keys 3, 5, 7, and 12.

NOTE: When editing within MemoPad, let MemoPad wrap long lines. Do not use newline until you are ready to begin a new instruction. Just let MemoPad warp the long line for you.

Sample program group created in MemoPad or on a PC should look like this:

```
##Trig

##Sine
##03
:computes sine of expr
x=expr(1)
sin(x)
```

This program is shown as part of a group named "Trig" and will be assigned to the 3rd button in the "Trig" group. This program will calculate the sine of any number entered into expression when the program is executed.

In APCalc the same program would look like this:

```
:computes sine of expr
x=expr(1)
sin(x)
```

The Function Key Button would be named 'Sine'.

Specifications

Floating Point	64 bit
Display Resolution	16 digits
Memory Req.	78k, plus space for saved programs
Functions	100 (73 APCalc, 27 MathLib) minimum; others to be added
Program Steps	64 steps
Expression Length	64 chars, 3 lines, Zoom Font
Storage Registers	100
Variables	30 total
History Buffer	32 expressions
User Keys	108, First 12 programs available on User 1 Function Keys
	Second group of 12 programs are on User 2 Function Keys
	Third group of 12 programs are on User 3 Function Keys
	Forth group of 12 programs are on User 4 Function Keys
	Fifth group of 12 programs are on User 5 Function Keys
	Etc.
	(All the Function Key Groups can be renamed)

Limitations (Unregistered Users)

Programs	3
Program steps	6
User Function Keys	3
Cannot Save New programs	

Registration

In order to have full use of all features of APCalc, registration is necessary. You can register by following the instructions on the registration screen. You access registration screen with: Menu→Info→How To Register

If you have registered APCalc, you should enter the Key that you received with your confirmation, into the Key field, and tap Register. User field and Reg. Code fields cannot be changed. Once the key has been recognized as valid, you will see the "Unregistered", at the top of the screen, change to "Registered" and a popup screen that says "Thank You". You will not be able to enter the key again, once you have successfully registered. Successful registration will remove this screen from being displayed each time APCalc is started.

Alternatively, you can tap Test Drive, without entering a Key, to use APCalc, in its demonstration mode, subject to the limitations of the demo version of APCalc   Demo mode limits the size and number of programs as well as the number of Function Keys.   If you have already registered, use Test Drive to exit the screen.

You may register at:

Handango
http://www.handango.com

Appendix A - License

The author of this software grants to the Licensee a non-exclusive, non-transferable, license to use this product in accordance with this agreement, on one device only. This software is supplied "as it", without warranty of any kind. The author assumes no liability for any damages incurred by its use.

The license policy is very simple. You are entitled to use the Demo version in the manner intended. That is, you are not entitled to use the extended features without registering the application. Registering the application gives you the right to use all features of the application, on one device. The use of the demo is limited to a reasonable time (15 days) to evaluate the program. After that time, you must register APCalc to continue with its use.

The following applies to all versions of this software: You may not reverse engineer, decompile, disassemble, modify, distribute, sell, give away, or post, this software, on any web site without written permission, of the author.

You may not use this software unless you agree to these terms. Using this software implies agreement with these terms.

Appendix B - Functions & Operators

Parameters used in the following functions use the following notation. dblReference, intReference and strReference. The prefixes 'dbl', 'int' and 'str' represent double, integer and string types. The 'Reference' part of the parameter is a descriptive label for the type of input parameter expected (i.e. dblAngle means an Angle, using type double, is expected as an input). String types must be surrounded by double quotes.

acos(dblValue)   Return type: double

 Returns the arc cosine, of dblValue, as a floating point value.   The return value is the angle expressed in either degrees or radians, depending on the Trig Mode selected.

acosh(dblValue)   Return type: double

 Returns the hyperbolic arc cosine, of dblValue, as a floating point value.   The return value is the angle expressed in either degrees or radians, depending on the Trig Mode selected.

and(intVal1, intVal2)   Return type: integer

 Returns the binary anding of intVal1 and intVal2. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than $2^{52}-1$.

asin(dblValue)   Return type: double

 Returns the arc sine, of dblValue, as a floating point value.   The return value is the angle expressed in either degrees or radians, depending on the Trig Mode selected.

asinh(dblValue)   Return type: double

 Returns the hyperbolic arc sine, of dblValue, as a floating point value.   The return value is the angle expressed in either degrees or radians, depending on the Trig Mode selected.

ask("strChoice0", "strChoice1", "strChoice2", "strChoice3", "strChoice4")   Return type: integer

 Displays a popup of list choices. Returns an integer value (0 - 5), depending on the selection made.   A maximum of 5 parameters (choices) may be used. You may use less than 5 choices.

atan2(dblY, dblX)   Return type: double

 Returns the arc tangent, of dblY/dblX, as a floating point value.   The return value is the angle expressed in either degrees or radians, depending on the Trig Mode selected.

atan(dblValue)   Return type: double

 Returns the arc tangent, of dblValue, as a floating point value.   The return value is the angle expressed in either degrees or radians, depending on the Trig Mode selected.

atanh(dblValue)   Return type: double

 Returns the hyperbolic arc tangent, of dblValue, as a floating point value.   The return value is the angle expressed in either degrees or radians, depending on the Trig Mode selected.

avg(intFromReg, intToReg)   Return type: double

 Returns the mean of all values from intFromReg and intToReg (inclusive). intFromReg and intToReg can be any values between 0 and 99. intToReg must be larger than intFromReg for this function to give correct results.

base(intBase)   Return type: integer

 Converts the last calculated value (Ans) to a new value and displays it, as intBase. intBase can be any integer between 2 and 36 inclusive.

beep(intFreq, intDur, intVol)   Return type: double

 Plays a sound of varying frequencies, duration and volume. intFreq is specified as hertz, intDur is specified in mili-seconds and intVol is an integer from 0 to 64. 0 is no sound. Returns Last Calculated Value (Ans).

BtB(intFromBase, intToBase)   Return type: integer

Convert the number in the expression field, in intFromBase format (i.e. if intFromBase is 10, then expression cannot contain non decimal values), to a number represented in intToBase. intFromBase and intToBase can be any integers between 2 and 36 inclusive. Note: Because this works on the expression field, it cannot be used outside of a program.

call("strProgram")   Return type: double

Executes a program call to strProgram. Up to 5 programs can be called before a rtn(1) or end of program (i.e. call stack is 5). Use of rtn(1) is optional. rtn(1) can be used to execute a program before the end of the program. This function returns the value of the last calculated value (Ans), at the time of the call instruction. It does not return a value as the result of the called program.

btn(mode)   Return type: double

Depending on value of mode, returns current Key Group (0), last function key pressed (1), program number (2).

callif(intIndex, "strProg0", "strProg1", "strProg2", "strProg3", "strProg4")   Return type: double

Executes a program based on an index. If index is 0, then strPgm0 is executed. Index is zero based. Parameters are optional except that at least one program must be specified. If index is blank or missing, callif() executes the same as call and uses strPgm0.

cbrt(dblValue)   Return type: double

Calculates the cube root of dblValue and returns it as a double.

ceil(dblValue)   Return type: integer

Computes the smallest integral value not less than dblValue.

clh(1)   Return type: double

Clear history buffer. Dummy parameter (1) required. Returns 'Ans'.

cls(1)   Return type: double

Clears the input expression field. Returns 'Ans'.

clv(1)   Return type: double

Deletes all variables. Dummy parameter (1) required. Returns 'Ans'.

comp(intVal1, intVal2)   Return type: integer

Compares two values. if intVal1 is larger than intVal2, a value of 1 is returned. If intVal2 is larger that intVal1, a value of -1 is returned. If the two values are equal, a 0 is returned. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than 2^52-1.

cos(dblAngle)   Return type: double

Returns the cosine, of dblAngle, as a floating point value. The dblAngle specified is in degrees or radians depending on the current Trig Mode.

cosh(dblAngle)   Return type: double

Returns the hyperbolic cosine, of dblAngle, as a floating point value. The dblAngle specified is in degrees or radians depending on the current Trig Mode.

di(intValue)   Return type: double

Set Display Instructions preference, 0 = don't display instructions, 1 = display instructions, 0 and 1 return the last calculated value returns current setting, -1 returns the current setting.

disp("strLabel")   Return type: double

Display a strLabel to the expression field. Returns the last calculated value (Ans).

dm(intValue)   Return type: double

Set Display Mode preference, 0 = Fixed, 1 = Scientific, 2=Engineering. 0 and 1 return the last calculated value, -1 = returns the current setting

dp(intValue)   Return type: double

Set Display Mode preference, 0 - 8 as an input, sets the number of decimal places.   0 to 8 return the last calculated value, -1 returns the current decimal point preference.

end(1)   Return type: double

Immediately stops program execution at this instruction.

exp(dblValue)   Return type: double

Calculates exponential of x (e^x).

expr(1)   Return type: double

Returns the numeric value of the expression in the input field. Use this as the first instruction to grab pending data. Us in the middle of a program, after a wait("prompt") to return the value of a newly input value or expression. You should use iff(pflag, "label") to test for the existence of pending data, before using it.

See pend(1)

fabs(dblValue)   Return type: double

Return the absolute value of dblValue.

fact(intValue)   Return type: integer

Calculates the factorial (dblValue!) of intValue.

fill(intFromReg, intToReg, dblValue)   Return type: double

Fills registers, starting with intFromReg and continuing to intToReg (inclusive) with the value dblValue. fill(0,99,0) would clear all registers. intToReg must be larger than intFromReg.

fk(intValue)   Return type: double

Set Function Key Mode preference, 0 - 13 as an input, and displays the current function key set.   0 to 13 return the last calculated value, -1 returns the current function key preference.

floor(dblValue)   Return type: integer

Returns the largest integral value not greater than dblValue.

fmod(dblValue1, dblValue2)   Return type: double

Computes the modulo remainder of dblValue1 divided by dblValue2.

frac(dblValue)   Return type: double

Returns the fractional part of a double value.

fv(intNP, dblIR, dblPMT, dblPV, intPT)   Return type: double

This function performs two functions. If there is new data input, the value is stored internally in the fv variable. If there is no new data pending the function returns the future value, of a time value of money calculation. intNP is entered as the number of periods in the loan. dblIR is the periodic interest rate. dblPMT is the amount of the payment (use - for payment and + for cash received). dblPV is the present value of the loan. intPT is zero for payments at end of the month and 1 for payments at the beginning of the month.

gaxis(xMin, xMax, yMin, yMax, step)   Return type: double

sets the grid spacing (x and y) for a function. This does nothing to specify the visual part of the axis. It is only used to scale the window and the function ranges. The last parameter is the value to use in stepping from xMin to xMax.

gcls(screen)   Return type: double
> clears graphic area. If 'screen' is 0, it will clear the entire screen, if 1 it
> will clear the current gwin() area.

gcolor(rfg, gfg, bfg, rbg, gbg, bbg)   Return type: double
> Sets the foreground and background colors that are used in other functions. r, g, b (red, green, blue), fg
> (foreground), bg (background).

gerr(intReportErrors)   Return type: double
> Reports error popup during graphing if intReportErrors != 0.   Reports errors if intReportErrors == 0.

gline(x1, y1, x2, y2)   Return type: double
> draw line from (x1,y1) to (x2,y2)

gplot("equation", "variable", drawAxis, drawConnected)   Return type: double
> use this function to pass a function, to graph, to the graphing program. You can specify an independant
> varialbe, other than x by using the second parameter. For instance: plot("sin(theta)", "theta"). drawAxis = 1
> to draw, 0 no draw. drawConnected = 1 draws between points. drawConnected = 0 draws only points.

gprt(x, y, "string")   Return type: double
> prints a "string" at coordinates (x,y)

gprt(x, y, val, decimalPlaces)   Return type: double
> prints a value at coordinates (x, y), using decimalPlaces as the number of decimal digits to display.

grect(type, x1, y1, W, H, fill)   Return type: double
> draw rectangle at (x1,y1) with Width W and height H. 'type' is frame width for non-filled rectangle and fill
> pattern for filled rectangle. 'type' for fill is 0 - fill with background, 1 - fill with foreground, 2 - fill with
> pattern. 'fill' is 0 for no fill and 1 for fill.

guln(mode)   Return type: double
> underline mode: 0 - no underline, 1 - dashed underline, 2 - solid underline, 3 - color underline.

gwin(x1, y1, W, H)   Return type: double
> specifies a drawing area for a graph or whatever you like. This is optional. If not used the window is draw
> at 10,10 and is 140 x 110. You can use this function to create multiple graphing windows.

hr(dblHMS)   Return type: double
> Returns time in decimal format, for time entered as a parameter in Degrees-Minutes-Seconds format. i.e. 1
> hour 30 minutes as a parameter returns 1.5 hours. dblHMS format is HH.MMSS.

hms(dblHR)   Return type: double
> Returns time in Degrees-Minutes-Seconds, for time entered as a parameter in decimal format. i.e. 1.5 hours
> as a parameter returns 1.30 (1 hour and 30 minutes).

hypot(dblX, dblY)   Return type: double
> Calculates the hypotenuse of right triangle.

iff(dblExpr, "strLabel")   Return type: integer
> Evaluates 'expr' and if TRUE (case not important), transfers program execution to strLabel. TRUE is any
> non-zero result. Zero is FALSE (case not important). Expressions can contain comparison operators such as
> x>3, etc. Returns the Last Calculated Value at the time the instruction is executed.

iff(dblExpr, expr1, expr2)   Return type: integer

Evaluates 'expr' and if TRUE, returns expr1 else returns expr2. 'expr1' and 'expr2' are both evaluated regardless of which is returned. For example you can use it in this way: iff(expr, x=3, y=4). If expr is TRUE 3 is returned, if expr is FALSE, 4 is returned. In both cases, x and y are assigned their respective values. TRUE is any non-zero result. Zero is FALSE. Expressions can contain comparison operators such as x>3, etc. This function requires 3 parameters.

in("strPrompt")   Return type: double

Pauses execution, displays an input field and asks for input (quotes required. Returns value input.

ir(intNP, dblIR, dblPV, dblFV, dblPMT, intPT)   Return type: double

This function performs two functions. If there is new data input, the value is stored internally in the ir variable. If there is no new data pending the function returns the periodic interest rate, of a time value of money calculation. intNP is entered as the number of periods in the loan. dblFV is the future value. dblPMT is the amount of the payment (use - for payment and + for cash received). dblPV is the present value of the loan. intPT is zero for payments at end of the month and 1 for payments at the beginning of the month. dblIR is initial guess use 1.

jmp("strLabel")   Return type: double

Absolute jump to strLabel. Note: Labels must start with : (ex. :loop1). Returns the last calculated value.

jmpif(intIndex, "strLbl0", "strLbl1", "strLbl2", "strLbl3", "strLbl4")   Return type: double

Absolute jump to label, depending on the value of index. If index is 0, then program execution transfers to strLbl0. Index is zero based. Parameters are optional. If index is blank or missing, jmpif executes the same as jmp and uses strLbl0. Returns the last calculated value.

ln(dblValue)   Return type: double

Computes the natural log of a double.

load("strProg")   Return type: double

Loads (chains) a new program. 'strProg' and Program name must match exactly. This function does not return to the calling program. Returns the last calculated value.

log(dblValue)   Return type: double

Computes log10 of a dblValue.

loop(intReg, "strLabel")   Return type: double

Program instructions between strLabel and the instruction loop() will be executed until intReg equals zero, at which time program execution continues with the instruction following the loop() instruction. intReg is the index register. Valid values, for intReg, are 0 to 99 inclusive. Load intReg prior to executing loop(), when loop() is executed, the intReg will be automatically decremented. The strLabel instruction must precede the loop() instruction. Returns last calculated value.

lset(intIndex, "strLabel1", "strLabel2", "strLabel3", "strLabel4", "strLabel5")   Return type: double

This label set function sets a group of five labels that will be displayed in the Registers Popup List. The function will return the Last Calculated Value. The first parameter specifies which label set is being set and the next five parameters are the labels. The first parameter, intIndex can be either 1 or 2. intIndex = 1 sets the first five labels and intIndex = 2 sets the next five labels. Only 10 labels can be set in this version and they will be contiguous. This will be changed in later versions to use a single function. This function in used in conjunction with regs() below. Label names can consist of up to 9 characters.

These labels are an array that will apply to any registers displayed with regs() with the second parameter set to 1. That means if you display registers 20-29 these labels will apply. If you display 40-49 the same labels will apply unless you change these. So you must change labels if you wish a different set for a different set of registers.

max(intVal1, intVal2)   Return type: double

    Returns the maximum of the two values. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than 2^52-1.

min(intVal1, intVal2)   Return type: double

    Returns the minimum of the two values. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than 2^52-1.

mtxf(intMatrix, dblValue)   Return type: double

    Fills a matrix (0 to 2) with the value of dblValue. Returns this value

mtxi(intMatrix, intRow, intCol, intStart)   Return type: double

    Initializes a new matrix (0 to 2). intRow specifies the number of rows. intCol specifies the number of cols. intStart specifies the starting register for the matrix. Return value is the sepcifics of the matrix in the following format: R.CST where R is number of rows, C is number of cols and ST is the starting register.

mtxg(intMatrix, intRow, intCol)   Return type: double

    Returns a specific matrix element. Specify the matrix (intMatrix), row and col.

mtxs(intMatrix, intRow, intCol, dblValue)   Return type: double

    Sets a specific matrix element. Specify the matrix, row and col.

mtxz(intMatrix)   Return type: double

    Returns the particular dimensions of the matrix as in mtxi but also saves these values in registers 97, 98 and 99.

nCr(intN, intR)   Return type: integer

    Returns combinations of intN objects taken intR at a time where intN and intR are integers.

not(intVal1)   Return type: integer

    Returns the one's complement of intVal1. intVal is an integer value. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than 2^52-1.

np(dblIR, dblPMT, dblFV, dblPV, intPT)   Return type: double

    This function performs two functions. If there is new data input, the value is stored internally in the np variable. If there is no new data pending the function returns the number of payment periods, of a time value of money calculation. intIR is entered as the periodic interest rate of the loan. dblFV is the future value. dblPMT is the amount of the payment (use - for payment and + for cash received). dblPV is the present value of the loan. intPT is zero for payments at end of the month and 1 for payments at the beginning of the month.

nPr(intN, intR)   Return type: integer

    Returns permutations of intN objects taken intR at a time. Where intN and intR are integer. Order matters.

or(intVal1, intVal2)   Return type: integer

    Returns the binary oring of intVal1 and intVal2. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than 2^52-1.

out("strLabel", dblExpr)   Return type: double

    Display a strLabel, in the expr field and a value in the answer field. Similar to disp except that it displays label and value, even when the display instructions' is set to no, in preferences. dblExpr can be any expression that returns a value. Returns dblExpr.

pend(1)   Return type: integer

    Similar to expr(1) except that it returns the pending data at the time the program was first executed, regardless of where it is used within a program.

pmt(intNP, dblIR, dblFV, dblPV, intPT) <div style="text-align: right;">Return type: integer</div>

This function performs two functions. If there is new data input, the value is stored internally in the pmt variable. If there is no new data pending the function returns the payment, of a time value of money calculation. intIR is entered as the periodic interest rate of the loan. dblFV is the future value. dblNP is the number of payments. dblPV is the present value of the loan. intPT is zero for payments at end of the month and 1 for payments at the beginning of the month.

Ex: 'ir' is the periodic interest rate (i.e. 9.9% with payments every month would be 9.9/12, as input for 'ir'). You may input the result of 9.9/12 or the expression itself.

np = total number of payments (i.e.: 360 for a 30 year mortgage)
ir = periodic interest rate (a yearly interest would be entered as xx/12)
pv = present value (ie. 70000)
fv = future value (ie. 0)
pmt = monthly payment (use – for a payment to someone else and + for a payment received).
pt = payment at beginning of month=1, end of month=0

pow(dblX, dblY) <div style="text-align: right;">Return type: double</div>

Computes dblX to the dblY power.

pv(intNP, dblIR, dblPMT, dblFV, intPT) <div style="text-align: right;">Return type: integer</div>

This function performs two functions. If there is new data input, the value is stored internally in the pmt variable. If there is no new data pending the function returns the payment, of a time value of money calculation. intIR is entered as the periodic interest rate of the loan. dblFV is the future value. dblNP is the number of payments. dblPMT is the loan payment amount. intPT is zero for payments at end of the month and 1 for payments at the beginning of the month.

rand(dblSeed, intMax) <div style="text-align: right;">Return type: double</div>

Returns a random number. The 2nd parameter is optional. If not used, the number returned is between 0 and 1. If you use intMax, the number returned is an integer format and is between 0 and intMax. intMax can be between 0 and 0x7FFF. If you wish to have a double returned that is between 0 and say 100, just use the function like this: rand(0)*100. The seed must be between $-1 < 0 < 1$.

rcl(intReg) <div style="text-align: right;">Return type: double</div>

Returns the value stored in register intReg. Valid values for intReg are 0 to 99 inclusive.

regs(intReg, intShowLabelsFlag) <div style="text-align: right;">Return type: double</div>

This function launches a popup list of the storage registers. The first item in the list is intReg. If the intShowLabelsFlag is set to 1, then labels will be used instead of register numbers. if intShowLabelsFlag is set to zero, then register numbers will be used. Within this 10 registers a ---- will be displayed if there are ther is no label and the contents of the register is zero. Labels are setup using set1 and set2 below. Tapping the register number will pop the register into the expression window. Tapping the register contents will select that register. Use lset() to specify the labels.

rint(dblValue) <div style="text-align: right;">Return type: integer</div>

Returns the integral value nearest dblValue in direction of prevailing rounding mode.

round(dblValue) <div style="text-align: right;">Return type: integer</div>

Round x to nearest integral value away from zero.

rtn(1) <div style="text-align: right;">Return type: double</div>

Return function. Optional use. End of a program also returns to calling program or ends a program, depending on whether or not it is a subroutine. Use this to exit from a routine before coming to the end of the routine (early exit).

rv(intValue) Return type: integer

Set Reset Variables preference, 0 = don't reset variables, 1 = reset variables, 0 and 1 return the last calculated value returns current setting, -1 returns the current setting. If RVars is checked, all variables will be reset each time the APCalc= key is tapped.

sdx(intFrom, intTo, intMode) Return type: double

Returns the standard deviation of a set of numbers stored in the built-in registers. Registers used are from intFrom to intTo. Legal values must be 0 to 99 for each parameter. The intMode parameter selects whether or not sample or population method is used to compute standard deviation. Use sample (intMode=0) when the data set represents a sample of the data rather than the entire population. Use population (intMode=1) when the data set represents the entire population. intMode is optional. If it is omitted, then sample method is used.

sin(dblAngle) Return type: double

Returns the sine, of dblAngle, as a floating point value. The dblAngle specified is in degrees or radians depending on the current Trig Mode.

sinh(dblAngle) Return type: double

Returns the hyperbolic sine, of dblAngle, as a floating point value. The dblAngle specified is in degrees or radians depending on the current Trig Mode.

slb(intVal1, intVal2) Return type: integer

Shifts intVal1 left by intVal2 bits. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than $2^{52}-1$.

sqrt(dblValue) Return type: double

Computes the square root of dblValue.

srb(intVal1, intVal2) Return type: integer

Shifts intVal1 right by intVal2 bits. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than $2^{52}-1$.

ss(intValue) Return type: integer

Set Single Step preference, 0 = no single-step, 1 = single-step, 0 and 1 return the last calculated value returns current setting, -1 returns the current setting. If single-step is checked, a program will require R/S to be pressed between instructions, to continue execution.

sto(intReg) Return type: double

Stores the last calculated value into register intReg. Valid values for intReg are 0 to 99 inclusive.

sum(intFromReg, intToReg) Return type: double

Returns the sum of all values from intFromReg to intToReg (inclusive). intFromReg and intToReg can be any values between 0 and 99. intToReg must be larger than intFromReg for this function to give correct results.

tan(dblAngle) Return type: double

Returns the tangent, of dblAngle, as a floating point value. The dblAngle specified is in degrees or radians depending on the current Trig Mode.

tanh(dblAngle) Return type: double

Returns the hyperbolic tangent, of dblAngle, as a floating point value. The dblAngle specified is in degrees or radians depending on the current Trig Mode.

tick(1) Return type: integer

Returns the number of clock ticks. This is a relative number. Compare one tick value to another to determine the number of ticks that have elapsed. Dummy parameter (1) required.

time(1)   Return type: integer

Returns the time to registers 93 to 99, in the following format.

Register 93   seconds
Register 94   minutes
Register 95   hours
Register 96   day
Register 97   month
Register 98   year
Register 99   days of week from Sunday (0 to 6)

tm(intValue)   Return type: double

Set Trig Mode preference, 0 = Degrees, 1 = Radians, 0 and 1 return the last calculated value returns current setting, -1 returns the current setting.

trunc(dblValue)   Return type: integer

Round x to nearest integral value not larger than x.

tz(intValue)   Return type: double

Set Trim Zeros preference, 0 = Don't trim, 1 = Trim Zeros, 0 and 1 return the last calculated value returns current setting, -1 returns the current setting

var("strVariable")   Return type: integer

Tests for the existence of a given variable, where "x" is any variable name. If the variable already exists a 1 is returned. If the variable is NEW, a 0 is returned. Use this in the iff() as follows: iff(var("x"), "calc-x").

vars(1)   Return type: double

Programmatically pops up the variable list in a modal form. Select variable that will be returned to the executing program.

wait("strLabel")   Return type: double

Pauses execution and displays a label. Label is string (quotes required). Returns 'Ans'.

xor(intVal1, intVal2)   Return type: integer

Returns the binary exclusive oring of intVal1 and intVal2. intVal1 and intVal2 are integer values. If you use floating point numbers, they will be truncated to the integer part of the value. Valid integers must be less than $2^{52}-1$.

xrt(dblX, dblY)   Return type: double

Calculate the x th root of y where x and y are doubles. Ex. 3rd root of 64 would be calculated with xrt(3,64) would evaluate to 4.

Constants

Constants represent fixed values. If you try to use a constant as a variable, you will get an error. For example, you cannot use: e=23 without generating an error. 'e' cannot be a variable.

pflag   Return type: N/A

pflag is actually a constant. The value of this variable is dependent on whether or not an expression is waiting to be processed (ie. APCalc= has not yet been pressed). pflag is TRUE (1), if an expression is pending and FALSE (0), if APCalc= has been pressed.

TRUE   Return type: integer

TRUE is actually a constant. The value of TRUE is 1. The case of TRUE (true) is not important.

FALSE   Return type: integer
>   pflag is actually a constant. The value of FALSE is 0. The case of FALSE (false) is not important.

e   Return type: couble
>   same as exp(1). NOTE: you cannot have a variable named 'e'. If you do, you will get an error since you cannot assign any value to this constant.

Ans   Return type: double
>   Last calculated value

FAns   Return type: integer
>   Last fixed point value

Special Keys

+/-   Return type: N/A
>   Changes the sign of whatever is in the expression window. Example: if sin(x)+cos(y) is in the expression field, tapping +/- changes the input to -(sin(x)+cos(y)). To change the sign of an individual value or variable, just use the '-' sign.

1/x   Return type: N/A
>   Computes the reciprocal of whatever is in the expression field. That is, if sin(x) is in the expression field, and you tap 1/x, you will see 1/(sin(x)).

Operators

+, -, *, /   Return type: double
>   Basic 4 function operators

:   Return type: double
>   Base conversion operator. Ans is expressed as the result of XX:YYYY, where YYYY is evaluated in base XX with the answer displayed in the current ToBase. See base conversion section.

%   Return type: double
>   Remainder operator. Returns the remainder of an division, in the form of XX%YY, where the remainder XX divided by YY is the decimal part of the result.

!   Return type: double
>   Computes the factorial of the number it is appended to. n! is equivalent to fact(n).

$ Return type: N/A
>   This is actually a line terminator rather than an operator. Repeats a multiple instruction line. The '$' must appear after a ';' Example: x?; x^2; iff(x==0, "end");$ NOTE: the '$' terminator is only meaningful, in an executing program. From the calculator level, one can simply just tap the APCalc= again for repeat operation.

NOTE: *Remember: all functions return a value. Some functions (such as sto()) return 'Ans' (last calculated value). That is: 2*sto(1) returns 2*Ans and also stores the previous 'Ans' to register 1. All functions that do not return a specific value as a function, behave this way.*

Appendix D - Operator Precedence

?	This simultaneously declares and forces prompting of a variable, even if the variable already exists. Use ? to initialize a variable at the beginning of a program.
^, ~, #,%	Raise to power, equality (~) for == , inequality (#) for !=, mod
*, /, \, ~, = =, **#, !=, <, <=, >,** **>=,**	Multiply, Divide, Integer Divide (trunc), Less Than, Greater Than # and != are equivalent. ~ and = = are equivalent.
+,-	Add, Subtract
=	Not really an operator. This is useful for assignments within a program. You an use assignments like: a=5, a=cos(x), a=cos(45), etc. ***Note:** the result of the expr is stored, not the expression. That is, a=cos(x) stores the value of cos(x), depending on value input; not the expression of cos(x). If you use var= without a value, the variable is deleted from the variable list.*

Operator precedence is from left to right within a precedence level.

`2*3^3*4`	Is the same as 2*(3^3)*4, which produces the result of 216.
`a*b<c*d`	Is the same as: ((a*b)<c)*d, if a=2, b=3, c=7, d=9 the result is nine. (2*3)<7 returns TRUE or (1), (1)*9 returns 9. If c were 5, the result would be 0.
`<, >`	Are useful within some equations and especially suited for 'iff' instruction.