RPN 3.62 Software Manual

Nth Lab

Table of Contents

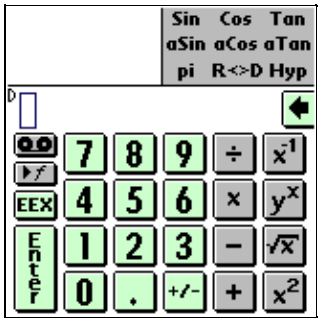
Table of Contents

Introduction

What is RPN?

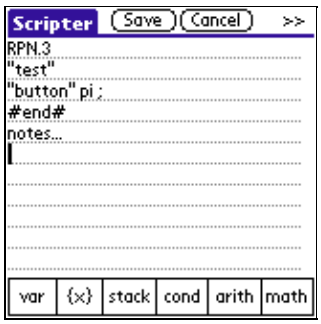
RPN is an advanced and highly customizable postfix calculator for scientific, engineering, and financial calculations. Please purchase RPN if you find it useful.

Major Features



- **Postfix**
- **Graphing**
- **Solving**
- **Programmable**
- **Recordable**

- **Hi−res interface**
- **Customizable colors**



- **Scientific and engineering functions**
- **Conversion functions**
- **Many format and base options**
- **Script editor**

- **No need for MathLib**
- **Palm OS 3.1 or later**
- **High precision**
- **100k in size**

Cost and Registration

There are two levels at which to purchase RPN:

- Standard − $29 − Free upgrades for life. Normal personal or professional use.
- Student − $19 − Free bug fixes. Student or infrequent hobby use.

RPN Coweb

The RPN Coweb is an online, structured discussion forum for all RPN customers to benefit from and contribute to. Click the link above to view the coweb.

Everyone can read the coweb contents. If you Join the Coweb, you can edit every page of content to help create the most adaptable and structured record of RPN information possible including uploading your own scripts.

Getting Started

Getting Oriented

Postfix Calculations

The basic idea of a postfix or revese polish notation calculator is that all operations take their arguments from a stack; which means, all numbers required by an operation are entered before the operation is selected. The stack is a storage area for numbers. The last value added to the stack is called the top of the stack (tos) and is the first item available when the stack is accessed. Note: RPN shows the stack building upwards from the input line.

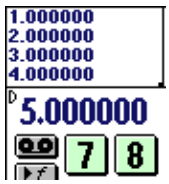
Rather than debate the benefits of this calculator style, I'll show a few examples. Note that operations "enter" the current input, so an 'enter' is not needed before an operation. (see the first example).

```
4+2 --> 4 enter 2 enter + --> 4 enter 2 +
(12+sin(7))/(11*3.1415) --> 7 sin 12 + 11 enter 3.1415 * /
```

Other Sources:
HP's RPN Tutorial

Unlike some postfix models (such as 4−level RPL by HP), in RPN what you see in the display always represents the top part of the stack. This model is more obvious to most people and was used in the HP−28 and HP−48 series of postfix calculators.

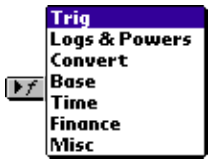
Stack and Input Area



The stack is displayed moving upwards in the upper left area of the screen. Just below the stack is the main input area (also to the left of the delete button). The input area is where you build up numerical input; when there is no numerical input the input area is used to display the first stack value.

Script Area and Button

The script area in the upper right corner of the display is fully customizable and dedicated to providing user input to RPN's powerful scripting language, RPN−Code. Each script controls this portion of the RPN screen to display a set of buttons which execute the RPN−Code defined for them. RPN includes a number of standard scripts to calculate trigonometric functions, logs &powers, unit conversions, etc. Users can easily create their own scripts or download scripts written by others.



You can scroll through the scripts using the scrolling keys on your device or pick a script from the popup list presented by tapping the script button which pops up a list of the installed scripts. Choose one and it will become the current script which is displayed in the upper right hand corner. Note, you can also scroll through the installed scripts using the page up and down keys on the device. See Built−In Scripts for an overview of each script that comes with RPN.

Recordings Button

RPN can record calculations you preform. Think of this feature like a 'watch and learn' way to build a calculation for later playback. Simply start the recording, perform the actions of your calculation, and then stop the recording. Recorded equations can then be played back, graphed, or solved (solving takes place inside the graphing dialog).

Record
: Begins recording all calculations, overwriting the last recording. Note, if you'd like to record a function that takes one input value, then you should put a value on the stack before recording since the input is not part of the function.

Stop
: Stop the current recording.

Play
: Playback the current recording.

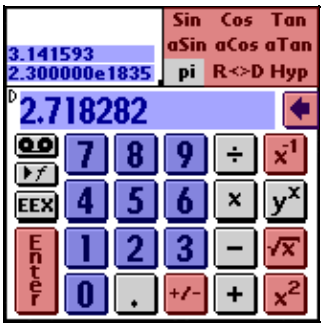
Graph/Solve
: Graph the current recording. Note, the recording needs to take one input and produce one output.

Save...
: Specify a name to save the current recording under (requires Palm OS 3.5 or later).

Load...
: Choose a previously saved recording to load.

Delete...
: Choose a saved recording to delete.

Drag and Drop



Dragging items around with the pen is a common shortcut to many RPN operations and in some cases the only way to use some features. To drag, tap a draggable item and, without lifting the stylus, drag that item to its destination and lift the stylus to drop. In the illustration, the blue areas are draggable items, while both the blue and red areas can be the destination of a dragging action. The most important operations are listed below (though you may discover others):

Store and recall of values
: To store a value, drag the value from the stack to one of the number buttons (0−9). Later you can drag the value back out of the number button and into any visible stack position.

Perform operations on values
: Drag a value (either from the stack or a storage location) and drop it onto any single argument operation (locations shown in red above).

Undo the last operation
: Drag the delete button to the main input area (just to the left).

Reorganize the stack
: Drag a value from one stack position to another. Hint: tapping a value will move it to the top of the stack.

Duplicate a value
: Drag a value to the 'Enter' button to copy it to the top of the stack.

Delete a value
: Drag the value from the stack to the 'Delete' button.

Menu Commands

There are several ways to get to the menus in RPN. You can use the menu button (usually located in or near the Graffiti area), the menu key on devices with keyboards, or **simply tap the top−left corner of the display** (it's easiest to tap all the way in the corner at the edge of the screen itself).

Drop All
: Clear the stack.

Last Stack
: Return the stack to previous state. Shortcut: Drag 'Delete' button to input area.

Drop
: (x −−>) Delete the top stack item. Shortcut: Drag value to 'del' button.

Swap
: (x y −−> y x) Move the 2nd stack item to the top. Shortcut: Tap the second stack item.

Dup
: (x −−> x x) Duplicate the top stack item. Shortcut: Tap the top stack item.

Over

(x y −−> x y x) Copy the 2nd item to the top of the stack. Shortcut: Drag value to 'Enter' button.

Rot

(x y z −−> y z x) Move the 3rd stack item to the top. Shortcut: Drag value to top of stack (input area).

New...

Open the script editor and start a new script.

Edit...

Open the script editor and load the RPN−Code for the current script. If you change the title of the script in the RPN−Code that changes which script you are editing. Note: If you defined the script in a previous version of RPN you will need to reinstall your source either by simply pasting your source into the script editor.

Delete...

Delete the current script.

Move(Front)

Move the current script to the beginning of the list of scripts.

Move(Back)

Move the current script to the end of the list of scripts.

Move(X)

Make the current script Xth in the list of scripts where X is the number on the stack.

Reinstall All...

Reinstall the default scripts; you will be asked to confirm any replacement of current scripts.

Key Shortcuts...

Define shortcut keys for common buttons using the "Key Shortcuts" dialog. All input mechanisms are treated the same so you can define Graffiti and keyboard based shortcuts using the same dialog.

Script Shortcuts...

Edit a special script (titled #keys#) that contains subroutines to be mapped to key shortcuts. The subroutines cannot reference each other or any global variables. For instance, to define a shortcut for the 'c' key you would put the following code into the script:
[c] code ;

About...

Displays information about RPN.

Help

Toggle the help mode ON and OFF. When help is on tapping a script command will display its help string though not all scripts have help strings.

Copy

Copy a single number from the stack to the clipboard.

Paste

Paste a single number from the clipboard onto the stack.

Modes...

Open the format modes dialog box.

Set Colors...

Set the colors used for the main display.

Using Graffiti or Keyboard

Most keyboard and Graffiti actions do the obvious thing. For instance entering '+' is the same as pressing the Addition button. The least obvious shortcut is that entering a space is the same as pressing the +/− button (this is because the Graffiti stroke for space looks like a minus).

Basic Calculation

Entering Numbers

Numbers can be entered using fixed, scientific, or engineering notation regardless of the current display mode. Pressing the EEX button will add the scientific part to the number, while the change sign button (+/−) will change the signing of whichever part of the number you are editing. Pressing the delete button while editing numbers deletes just the last digit of the number.

Pressing the EEX button to start a new number will automatically begin the number with "1e", which simplifies entering numbers like 1e6.

Formatting Numbers

Choosing Modes... from the Extra menu or tapping the left side of the input line (around the trig mode display) will bring up the following dialog where you can define how numbers are formatted for display:

Trig Mode
: Which units are used for degrees.

Comma Mode
: Which character to use for the comma separator. The decimal separator will adjust automatically.

Number Mode
: Which format to use for numbers.

Fixed	Scientific	Engineering
10,000.12	1.00012e4	10.00012e3

Places
: How many places to show after the decimal point. Tapping Places allows you to choose from common options.

Base
: What numerical base to use for numbers. Tapping Base allows you to choose from common options.

Show Trailing Zeros
: Set whether extra zeros after the decimal point are shown. (example: 1.0100 vs. 1.01 both with Places set to 4)

Performing Calculations

Calculations are preformed by pressing buttons in the main display, choosing scripts in the script area, or entering key shortcuts via Graffiti or the keyboard. All calculations happen immediately and operate on the currently visible stack. In this way, postfix calculation is more immediate and separate from numerical data than infix or algebraic computation.

Built−In Scripts

The built−in scripts are installed when you frist run RPN and appear in the upper−right of the RPN display. You can page through scripts with the scroll buttons on your handheld or select a script using the script popup button, the button with a little triangle on it (just above the Enter button). The usage of each built−in script is described below (only the more complex operations are covered in detail):

Trig

A collection of trigonometric functions and operations.

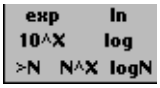
RD

(deg −−> deg*)
Presents you with a dialog to choose between converting to radians, degrees, or into the current mode of the RPN display.

Hyp

(x −−> f(x))
Choose between various hyperbolic trig functions.

Logs &Powers



Logorithmic and exponential operations.

>N

(N −−>)
Set the value for N used in the next two functions.

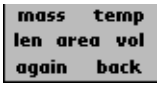
N^X

(X −−> N^X)
N to the power of X

logN

(X −−> logN(X))
Take the base N logarithm of X.

Convert



Over 400 conversions of various units.

mass, temp, len, area, vol

(unit1 −−> unit2)
Choose two units and a conversion from the first into the second is made.

again

(unit1 −−> unit2)

Repeat the last selected conversion.

back

(unit1 −−> unit2)
Reverse the last selected conversion.

Base

Do calculations in different numerical bases.

>B

(base −−>)
Set the displayed base to any value in [2, 32] (the display can get a bit confusing in bases greater than 16 if you aren't used to it): 2 = binary, 8 = octal, 16 = hex.

+, −, *, /, ,|, ^

(Y X −−> b(Y,X))
Basic binary operations.

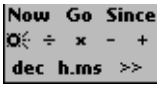
<, >, ~

(X −−> u(X))
Basic unary operations.

A−F

(−−>)
Enter Hex digits A−F if you are in a base which supports them (greater than base 10 obviously).

Time



With this script you can do arithmetic on time values and use clock and stopwatch features. All calculations are done in *h.mmsstt* format; meaning time is packed into parts of a decimal number so that 12 hours, 4 minutes, 29 seconds, and 30 ticks (1/100th sec) would read 12.042930.

Now

(−−> current_time)
Report the current time.

Go

(−−>)
Start the stopwatch.

Since

(−−> elapsed)
Report the elapsed time since the last Start without stopping the stopwatch.

(h.mmss −−>)

Set a countdown alarm. The alarm can be Off, Running &Shown, or Running &Hidden. If the handheld is on and this script is visible then an alarm will go off when the timer completes. Note: this timer will not turn the haldheld on when it goes off, but it will continue when you come back to the Time script even if you change applications.

/, x

(timeY X −−> b(timeY,X))
Divide or multiply in time format. Note: X is a regular number not a time.

−, +

(timeY timeX −−> b(timeY,timeX))
Subtract or add in time format.

dec

(h.mmss −−> decimal_time)
Convert from h.mmss to decimal time format.

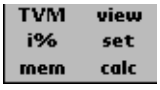
h.ms

(decimal_time −−> h.mmss)
Convert from decimal to h.mmss time format.

>>

(time −−> shifted_time)
Shift decimal places 2 places to the right.
For instance, to enter 47 seconds: *tap(4, 7, >>, >>)*

Finance



Time value of money (TVM) calculations are done using 5 main variables: present value of money (pv), effective interest rate (i), payment amount (pmt), number of periods (n), and future value of money (fv). If you know 4 of these values, then this script allows you to solve for the 5th one.

Note: The interest period and payment period is the same in these calculations to simplify the interface; you will have to convert your payments and interest into the same period.

Note: If the interest rate is zero then you should just use the obvious calculations. Solving for extremely small interest rates (below .0001%) is not supported in this version.

Note: Interest is entered and reported in decimal form, so that 6% interest is 0.06

Note: As usual in these calculations, money you have is positive and money you don't have is negative.

Short Example:
Let's say you want to invest $1,200/year for 10 years and need $40,000 at the end of that time to build your very own robotic pirate. How much interest are

you going to need to earn on your investment?
tap(mem, clear), −1200 *tap(set, pmt)*, 10 *tap(set, n)*, 40000 tap(set, fv), *tap(calc, i)* = 25% interest/yr!
Let's say you're willing to start with $10,000.
−10000 *tap(set, pv)*, *tap(calc, i)* = 8.3% interest/yr. Now that's a more likely way to acheive your goal.

TVM

(−−>)
Set the modes of TVM calculations using dialogs. You can specify when payments are made in the period and whether calculated values automatically are set after calculation.

i%

(i −−> i*)
Convert between Nominal and Effective interest.

mem

(−−>)
Save, recall or clear all the TVM settings and values.

view

(−−> varies)
View one or all of the 5 stored values.

set

(varies −−>)
Set one or all of the 5 stored values. To set all you need to put them on the stack in the order listed in the 'set' dialog.

calc

(−−> result)
Calculate one of the 5 TVM values using the other 4 stored values.

Misc

Miscellaneous functions.

mod

(Y X −−> Y%X)
Remainder after dividing Y by X.

ip

(X −−> ip(X))
Integer part of X.

fp

(X −−> fp(X))
Fractional part of X.

!

(X −−> !X)
Factorial of X.

%

(Y X −−> Y Y(X/100))
Take X% of Y, leaving Y on the stack.

%%

(Y X −−> delta(Y,X))
Percent change of Y becoming X.

stats

(... −−> stdDev mean sum N)
Calculate statistics about the list of numbers on the stack.

Undoing Actions

You can undo the last actions effect on the stack by choosing Last Stack from the Stack menu or by dragging the 'Delete' button to the input line (which is directly to the left of the 'Delete' button). Note: effects on variables in scripts are not undone.

Advanced Calculation

Recordings

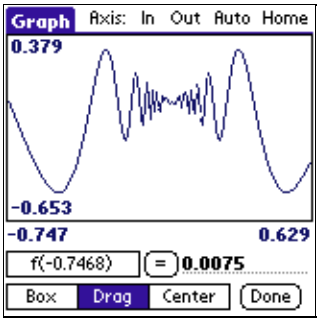
Also see recordings button and the next section for a Graphing discussion.

The sequence of actions to make a recording for the function f(x) = x*x + 7 is given below:

1. Enter a starting value and press enter. Note, the starting value is not part of the recording since you want the recording to operate on any value from the stack.
2. Select 'Record' from the 'Recordings Button'.
3. Press 'Enter' to duplicate the value.
4. Press the 'Multiply' button.
5. Press the '7' button.
6. Press the 'Add' button.
7. Select 'Stop' from the 'Recording Button'.

Graphing

Once you have recorded a function that produces a value for Y (see above), RPN will graph and solve it by supplying values for X and then runing your recording. Select 'Graph/Solve' from the 'Recordings Button' to view the graph. To return to the main calculator display simply tap the 'Done' button.



Axis Tools

Found along the top of the display, the Axis Tools allow you to adjust the graph view with a single tap:

Axis:
Manually set the bounds.

In
Zoom−In by 2x.

Out
Zoom−Out by 2x.

Pen Tools

At the bottom of the display, the currently selected Pen Tool determines what happens when the pen is drawn over the graph:

Probe − f(x)
Tracks the pen in the graph and displays values.

Box
Drag out a rectangle and the graph will zoom−in to that view.

Auto
Adjust the view so the graph fits.

Home
Go to the home view [−1, 1].

Drag
Drag the pen to move the graph within the view.

Center
Choose a point on the graph to center in the view.

Solving

The probe tool displayed beneath the graph is used for solving as well as the reporting values. When the = button is pressed, RPN attempts to find an x value for the function graphed that equals the number in the field to the right of the button.

Within the graph display you can solve for the x−value corresponding to a target y−value:

1. Display a portion of the graph where the desired solution exists.
2. Allowing RPN to graph all the points may increase the success of solving.
3. Write the desired value in the field to the right of the = button.
4. Press the = button.
5. If valid starting conditions are found, RPN will display the result to the left of the = button.

Scripts

What are scripts?

RPN is programmed in a language called RPN−Code. Each script presents its own interface (the buttons in the upper−right in RPN) and is defined by loading RPN−Code written using the built−in script editor or an external editor.

Navigating Installed Scripts

There are three ways to navigate between different scripts:

- Scrolling through them with the scroll keys on the device.
- Selecting one from the Script Button popup.
- Activating a script that executes a navigation action. This is an advanced topic covered in the programming section, see Misc Bytecodes.

Finding and Installing Scripts

The best place to find scripts is the RPN Coweb. Once you've found a useful script database simply hotsync it to your device. The next time RPN is run it will be imported. In order to see that it is installed you may need to navigate to it.

Managing Scripts

You can change the order of scripts and delete unwanted scripts using the Script Menu.

Basics of Scripting

Introduction to RPN−Code

RPN is an unusual language in that much of the code you write is actually the bytecode that is executed by RPN. All RPN bytecode is made up of typeable ASCII characters (for instance the bytecode for addition is '+').

All scripts contain structure such as the header, title, and labels, which is not RPN−Code. The RPN−Code is all the executable code in a script, and there are two supported modes of writing RPN−Code. These modes are selected by the script header which should start with either "RPN.2" or "RPN.4". RPN.3 has been dropped because it is incompatible with and harder to use than RPN.4. If you need to migrate code from RPN.3 format to RPN.4 see Migrating from RPN.3

In RPN.2 form, all code your write is bytecode and spaces are stripped out of your code.

In RPN.4 form, code is broken into *words* based solely on whitespace, so "4+2" is one word with a length of 3; while, " 4 + 2 " is 3 words each with a length of one. These words are then executed as described below (the first match from the list is used):

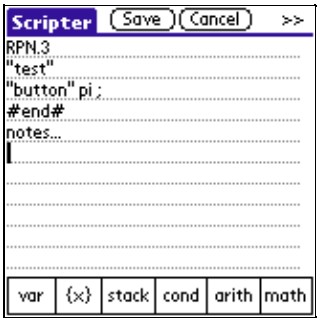
	Name	Action	Format
1.	local	(−− value) get local	[a−z|A−Z|0−9|_] not starting with a digit. Like C identifiers.
2.	subroutine	(? −− ?) call subroutine	any characters except: "()'[]`{};
3.	=local	(value −−) set local	=[local format]
4.	number	(−− value) push number	123, 2e−4, .3, ...
5.	command	(? −− ?) execute command	fixed
6.	bcode	(? −− ?) run string as bytecode	any

RPN has a simple trace utility to aid in script debugging. Executing the code, 'Ut', toggles the trace state. Pressing the Record button while a function is running will also activate the trace utility and allow the function to be traced, aborted, or continued. The stack gauge (between the stack and function display) is updated periodically while functions are running.

When an illegal action is taken in RPN−Code the function which is running aborts, and a message is displayed to the user indicating the reason for the error.

Editing Scripts

To edit the currently visible script choose Edit... from the Script menu (to make the menus visible tap the upper left corner of the screen or use your device's built−in menu button or key). If the RPN−Code for the script is available, the editor will be opened and the code loaded. Scripts installed and migrated from before RPN 3.50 will not have RPN−Code installed and you will need to paste their source in from its original source. The edit form and its parts are detailed below:



Script Editor Title
: Tap here to access the Edit menu.

Save Button
: Save changes to the script.

Cancel Button
: Tap here to discard changes without loading the script for use.

Next Placeholder Button (>>)
: Move cursor to the next place holder. The tab key or stroke also does this.

Code Area
: The code for the script is edited here.

Token Popup − var
: Insert available subroutine and variable names into the script.

Structure Popup − {x}
: Insert various programming structures with placeholders for input.

Stack Popup − stack
: Insert stack commands into script.

Query Popup − cond
: Insert query commands into script.

Arithmatic Popup − arith
: Insert basic math commands into script.

Math Popup − math
: Insert math commands into script.

Make any changes you wish to the script and then press Save to load the script into RPN for use. If you wish to create a new script from an old one, simply change the title in the RPN−Code and a new script will be created when you press Save, leaving the old script unmodified.

Exporting Scripts

You can use the Export menu item to create an exportable database of your script. After selecting Export, you will be asked which export database to place the script into. This allows you to make a package containing several scripts that are to be distributed together.

Once the export is created and after hotsync, you will find a database in your backup folder named "Yrpn_MyExportName.pdb". You can rename this file however you wish provided you leave the ".pdb" extension.

The best place to distribute your exported scripts is at the RPN Coweb.

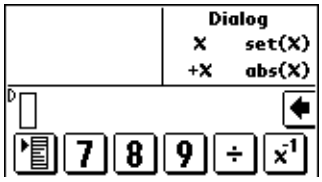
Programming Tutorial

The following tutorial will walk through an example of creating a script. The steps for inputing the script are as follows:

1. Start the script editor using the Edit... item in the Script Menu.
2. Enter the script text (delete the starting text if needed).
3. Tap the Save button.

Structure of a Script

The structure of a script is explained below using an example script:




```
0 RPN.4.a
1 [dialog] D'hello world' ;
2 [a]xa; [=a]Xa;
3 "Example"
4 "Hello" dialog ;
5 ~
6 "X" a ; \ recall 'a' to tos
7 "_set(X)" =a ;
8 ~
9 "_+X" a + ;
```

```
10 "abs(X)" a 0<( a neg : a ) ;
```

0	Script header: RPN.2 or RPN.4 is required for all scripts; '.a' creates one global variable.
1	Subroutine: just below the script header any number of subroutines can be defined; each is labeled with a name inside brackets. This subroutine opens a dialog displaying 'a subroutine'.
2	Subroutines: Define subroutines to access the global variable.
3	Title: the script name which will be displayed in the script popup menu. The title is the first label in double quotes in the script.
4	Button: buttons are defined from left to right and top to bottom in the script button area; the code after the label will be executed when the user taps the button.
5	New line: buttons defined after the ~ will be on the line below previous buttons in the script area.
6	Button: this button will put the variable 'a' on the stack.
7	Button: this button will set the variable 'a'; the leading underscore character allows the user to drag values to the button and should be used for all buttons taking exactly one value off the stack. The underscore is not displayed as part of the name.
8	New line: start the third line of buttons in the script.
9	Button: this button puts the variable 'a' on the stack and adds.
10	Button: a slightly more complex button script: `if a < 0 then neg(a) else a end`

Basic Techniques

Button Layout

Each script presents a collection of buttons for the user to press. See the Programming Tutorial above for an example.

Subroutines

Subroutines are an important part of writing readable code. After the header of each script you can define a list of subroutines for that script and then call the subroutines by name in any part of your RPN−Code.

Local Variables

Each subroutine and button has its own local variables which are not persistent. The is one variable aways available; it is read and written by the RPN−Code bytecodes: 'v' and 'V'.

In addition names locals can be allocated and used in RPN.4 code. These use a higher level word based syntax to encourage their use:

action	stack effect	RPN−Code
create or set variable named "x"	(tos −)	=x
get value of variable named "x"	(− x)	x

Global Variables

Each script has 0−255 global variables. The number defined is determined by the script header. For instance, starting a script with "RPN.4.c" defines three global variables ('a' to 'c'). These variables can be accessed from anywhere in the script and are presistent and even backed−up when you sync your device.

To read the 'b' variable onto the stack, you use the RPN−Code 'xb'. To write from the stack to the variable, use the RPN−Code: 'Xb'.

To define more than 26 globals you have to use RPN.4 form and define them in groups of 10 using a capital letter in header. You can access them using the X@ and x@ bcodes. So RPN.4.C will define 30 variables (the first 26 can still be accessed with x[a−z]).

Conditionals

There is one main conditional structure in RPN−Code:

```
( true_code : false_code )
```

When such a conditional is executed a boolean value should be on the top of the stack. Either the true or the false section of code is executed but never both. Conditionals can of course be nested, and the false code section including the ':' is optional. Example (which negates tos if it is negative and adds 4 otherwise):

```
dup 0 < (n : 4+)
```

The other conditional related RPN−Code can be found in the RPN−Code Reference.

Loops

There is one main looping structure in RPN−Code:

```
{ loop_code }
```

All loops are infinite loops and must contain code to break out of the loop. The basic break code is 'B'. Example (which simply breaks out of the loop using the break code):

```
{ 1 2 < (B) }
```

Another useful construct is a count down loop:

```
9V{ code _v} / loop 10 times 9 -> 0
```

The other looping related RPN−Code can be found in the RPN−Code Reference.

Script Format

The numbers to the left match the notes below and are not part of RPN−Code:

```
1.      RPN.version[.variables[+|−width]]
2.      subroutines
3.      "title"
4.      "label1" code ;
5.      "_label2: help" code ;
6.      ~
7.      \comment\
8.      \whole line comment
9.      "label3" code ; "label4" code ;
        ...
```

1. The first five characters of any RPN script must be of the form RPN.X[.g[(+|−)w]] where X is the RPN−Code version number that the script requires. Currently the version for new script should always be '2' or '4'. The variable field of the header indicates the number of global variables and is discussed in the section on global variables. The width field of the header, w, indicates how to adjust the width of the script window. Valid widths are −8 to 8. The width of the window is adjusted by 10 pixels per increment, so RPN.2.a+5 makes the script area 40 pixels wider than its usual 80 pixels. The variable and width field are optional. If they both are present the variable field must come first.
2. Subroutines are of the following form: [name] code ;
   Subroutines named with a single letter can be called by using the RPN−Code, C. Other subroutines are called by name if using RPN.4. Subroutines must come after the "RPN.X" and before the first label.
3. Labels are formed by enclosing text in quotes. The first label is the title of the script being defined. "" is not a valid label character.
4. Defining a button is done by specifying a label followed by the button's RPN−Code. All button definitions must end in a semi−colon.
5. Starting a function label with the '_' character allows values to be dragged to that button. The '_' character is not displayed and the function should have the following stack signature: (n −−> ?). The ':' character signals the beginning of the button's help string. The ':' and following characters are not

displayed. When help is turned on the entire button label (3 lines maximum) is shown in a RPN dialog (use '\' for a line feed);
6. The '~' character is used to separate lines of buttons. There can be up to 4 lines of buttons defined by any script. All lines have the same height, and all buttons on a row are of the same width. This height and width is adjusted according to the number of lines and buttons.
7. The '\' character is used to delimit comments.
8. The end of a line also ends a comment.

Note: There must be an even number of ' and " characters in every valid script; otherwise, error checking will reject the script.

Example:

```
RPN.1
[p]#'3.1415';
"Example"
~
"+" +;
"–" –;
"*" *;
"/" /;
"%\mod" %;
~
"_sin" i;
"_cos" o;
"_tan" a;
~
"_exp" e;
"_ln" l;
"2pi" 2Cp*;
```

Commands

RPN−Code can contain special commands words rather than bytecode. While making your code longer (usually) these commands make your code more readable. Unlike bytecodes, all of these commands need to be delimited by whitespace on both sides.

	Command	Stack Effect	Bytecode
Stack			
	dup	(x −− x x)	g1
	drop	(x −−)	d1
	swap	(y x −− x y)	r2
	rot	(z y x −− y x z)	r3
	nip	(y x −− x)	r2d1
	tuck	(z y x −− x z y)	k3
	over	(y x −− y x y)	g2

	store	(stack −− stack)	Z
	recall	(−− previous_stack)	z
Control			
	break	(−−)	B
	==	(y x −− f)	=0
	>=	(y x −− f)	g2g2>k3=0|
	<=	(y x −− f)	g2g2<k3=0|
	!=	(y x −− f)	=0!
Math			
	err	(−− numerical_error)	E
	pi	(−− pi)	#'3.1415926535897932'
	abs	(x −− abs(x))	b
	neg	(x −− −x)	n
	fp	(x −− fractional_part(x))	f
	wp	(x −− whole_part(x))	w
	inv	(x −− 1/x)	t
	pow	(y x −− y^x)	P
	exp	(x −− e^x)	e
	ln	(x −− ln(x))	l
	log	(x −− log10(x))	L
	sqrt	(x −− squareroot(x))	s
Trig			
	sin	(x −− sin(x))	i
	cos	(x −− cos(x))	o
	tan	(x −− tan(x))	a
	asin	(x −− asin(x))	I
	acos	(x −− acos(x))	O
	atan	(x −− atan(x))	A
	rad	(x_in_rad −− trig_mode(x_in_rad))	Mr!(180 pi /*)
	deg	(x_in_deg −− trig_mode(x_in_deg))	Mr(pi 180 /*)

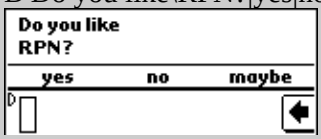
Basic Bytecodes

The following RPN−Code list describes the basic operations which are most commonly used in scripting RPN. There is a complete list in the next section for all RPN−Code.

Stack:

In the Stack column, the number of items removed from the stack and the number of items put onto the stack are listed separated by the »character.

Format	Name	Stack	Notes
0−9	number	0 » 1	push 0−9 onto stack
#'xxx'	literal	0 » 1	convert string to number using base 10 and push the value onto the stack
;	exit	0 » 0	exit subroutine
V	setVar	1 » 0	set local variable; this variable is unique to each subroutine (ie. each subroutine has its own value of v
v	getVar	0 » 1	push the subroutine's local variable onto the stack
+ − * /	basics	2 » 1	add, subtract, multiply, divide; removes two values from the stack and return result
P	power	2 » 1	returns nos**tos
b	abs	1 » 1	absolute value
n	negate	1 » 1	negate
g1	dup	1 » 2	duplicate tos
d1	drop	1 » 0	remove tos from the stack
r2	swap	2 » 2	swap the top two stack items
h	depth	0 » 1	puts the stack depth on the stack (1 2 3 −−> 1 2 3 3)
&| ^ >	boolean operations	2 » 1	logical and bitwise binary operations
!	boolean not	1 » 1	converts true to false, and false to true
=X	equal	2 » 1	returns true if |nos−tos| '=0' does an exact comparison
(	if	1 » 0	start conditional
:	else	0 » 0	separate conditional code; mark end of code to execute when true and beginning of code to execute when false; else is not required
)	endif	0 » 0	close conditional
{	begin	0 » 0	begin loop
}	repeat	0 » 0	repeat loop
B	break	0 » 0	leave current loop

D'text'	dialog	0 » 1	show dialog and report button pressed; D'Do you like\RPN?|yes|no|maybe|' creates 
CX	call	0 » 0	call subroutine labeled [x]

Advanced Scripting

Advanced programming topics are covered here. If these details do not make sense, then you probably don't need them.

Migrating from RPN.3

Unfortunately, with the introduction of RPN.4 form of RPN−Code, the RPN.3 form had to be dropped. Many of your RPN.3 scripts will work without modification. Below is the list of changes you will need to make sure your RPN.3 code can be used in future versions of RPN:

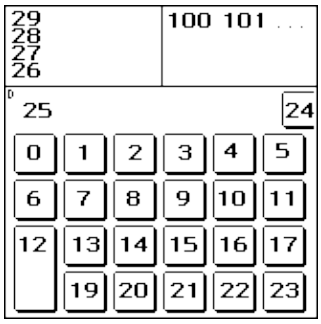
1. Change the header to be RPN.4 rather than RPN.3.
2. Wherever you used the old local.make or local.set syntax replace both =local. Currently .set and .make still work but they will be removed soon.
3. RPN.3 allowed you to access globals by single character references. RPN.4 does not. Simply define a subroutine for each variable: [a]xa; and that will allow the same code to work, though you might consider naming your globals something better than a−z.
4. Subroutines are now searched before locals, this is so that [=global]Xa; will be found instead of setting a local.
5. Locals and subroutines have better defined naming limitations (see the Formats in the Introduction to RPN−Code).

Event Handlers

RPN includes special subroutine forms for intercepting and handling events. When event handlers are called RPN does not enter the user's input or update the display after the handler is done; the handler should use 'Ue' and 'Ud' to cause an enter and display action to occur. Handlers which override the a default behavior (such as a key or button action) of RPN should use 'Uh', handled, to signal that the event has been handled. If the handler does not execute 'Uh' then RPN will handle the event as usual. The handler subroutines have the form {x}ByteCodes; where x determines the event to handle as follows:

```
x=k --> key    (ascii  --> ?)
x=b --> button (number --> ?)
x=t --> timed  ( --> ?)
x=o --> open   ( --> ?)  should not interfere with scrolling through scripts
x=c --> close  ( --> ?)  should not interfere with scrolling through scripts
```

For the button handler, {b}, buttons are numbered as follows:



Bytecode Reference

All the bytecodes available for RPN scripting are detailed here:

[Literal | Boolean | Control | Variables | Stack | Math | Trig | Misc]

Stack Notation:
Each byte code is described in stack notation having the form, "before −−> after." This notation documents the required arguments and the results of the byte code. All byte codes consume their arguments, so "x y −−> x+y" means x and y are popped off the stack and x+y pushed on to the stack. The shorthand tos, nos, and flag are used for TopOfStack, NextOnStack, and boolean values. The table of byte codes also includes links to further information on some byte codes.

Literal

[**Literal** | Boolean | Control | Variables | Stack | Math | Trig | Misc]

Format	Name	Stack Description	Notes
0−9	number	−−> value	push 0−9 onto stack
#'xxx'	literal	−−> xxx	convert string to number using base 10; #" gets a random, signed 32−bit number

Boolean

[Literal | **Boolean** | Control | Variables | Stack | Math | Trig | Misc]

Boolean values [(−1,1)=false, other=true] are produced by comparison operations and can be combined using logical operators. Only the whole part of the number is used to determine its logical value, so the

non−inclusive range (−1,1) is false. In general, boolean values should not be mixed with numerical values and should not be left on the stack; however, it may be of use to know that the logical operators (|^)are actually bitwise operators as long as their inputs fit into a unsigned 32−bit number. Negative numbers can be used as boolean values but only their absolute value matters; logical operations return unsigned numbers regardless of input signs. The not code, !, is logical rather than bitwise. If you find the above information confusing, please just think of the boolean codes as operating on the special values returned by comparison (ie. you don't need to think about what the numerical value of booleans are unless you combine them with numbers, display them, or use a number as a boolean).

Format	Name	Stack Description	Notes
!	not	flag −−> !flag	negate a boolean (logical negation)
&	and	flag1 flag2 −−> flag1	AND together two booleans (limited range bitwise)
|	or	flag1 flag2 −−> flag1|flag2	OR together two booleans (limited range bitwise)
^	xor	flag1 flag2 −−> flag1^flag2	XOR together two booleans (limited range bitwise)
=X	equal	nos tos −−> flag	returns true if |nos−tos| use '=@' to get X from the stack; '=0' does an exact comparison
<	less	nos tos −−> flag	is nos less than tos?
>	greater	nos tos −−> flag	is nos greater than tos?

Control

[Literal | Boolean | **Control** | Variables | Stack | Math | Trig | Misc]

RPN's control codes allow you to setup conditional and looping structures. Both can be nested (to greater than 100 levels).

Format	Name	Stack Description	Notes
(	if	flag −−>	start conditional
:	else	−−>	separate conditional code; mark end of code to execute when true and beginning of code to execute when false; else is not required
)	endif	−−>	close conditional
{	begin	−−>	begin loop
}	repeat	−−>	repeat loop
B	break	−−>	leave current loop
]	continue	−−>	

			goto the beginning of the current loop
_[x[a−z]|v]	var break	−−>	decrement indicated variable (see section on variables) and break if the variable is negative after decrementing (note: '_x@' is not valid)
R	restart	−−>	restart the current routine
CX	call	−−>	call subroutine labeled [x]
;	exit	−−>	exit function
.	abort	−−>	silently abort the current program
c	choose	tos −−>	a switch statement; format: "c(0−:1:2:3:4+)" where the numbers indicate the code that tos selects; note the last block is executed by by large inputs and the first block is executed by inputs less than or equal to zero; the skip command can be used to implement ranges: "c(0−:1:,:2|3:4+)"
,	skip	−−>	skip the next byte; "1,23+" == 4

Variables

[Literal | Boolean | Control | **Variables** | Stack | Math | Trig | Misc]

There are two kinds of variables in RPN which are accessed with low−level RPN−Code.

One local variable is available to all buttons and subroutines. This variable is private to each routine and is accessed using the codes 'v' and 'V' (they are volatile hence the naming).

The local variable is always available and should be used before globals unless the value needs to be persistent. Using it can speed your code, use less system resources, simplify your global variable space, and make your code less affected by later changes in RPN.

Global persistent variables are also provided by RPN. Global variables are stored with your function set and and cannot be accessed outside your set (but they are global within the set). Globals are declared in the RPN−Code header. Specifying RPN.3.g, indicates that this code needs global variables a−g to be created and managed by RPN. Function sets can be created with no globals by leaving off the global specifier (ie. RPN.3). These globals are accessed using the 'x' and 'X' codes (see below). Accessing an undefined variable will abort your function. When the Functions|Replace menu command is used to overwrite a set having the same name as the set being installed, the global variables defined in both sets are copied from the old to the new set. If you replace a script, global variables are copied over from the previous installation.

Globals can also be accessed dynamically using 'x@' and 'X@' where an index to the globals is on the stack. This is a one−based array (ie. 1x@ == xa).

Format	Name	Stack Description	Notes
x[a−z|@]	get var	−−> value	get global variable
X[a−z|@]	set var	value −−>	set global variable
v	get var	−−> value	get local variable
V	set var	value −−>	set local variable

Misc

[Literal | Boolean | Control | Variables | Stack | Math | Trig | **Misc**]

Format	Name	Stack Description	Notes
E	error	−−> value	put error value on stack; same result as "10/"
MX	Mode	−−> Mode_Value	X indicates mode to get; r=radianFlag; v=rpnVersionNumber; other modes may be defined later
G'xxx'	goto function set	−−>	goto (load) the function set named xxx
U[t|d|e|b|o|h|T]	user interface action	see notes	**opcode** / **name** / **action** t / trace / −−> toggle byte code tracing mode d / display / −−> redraw the RPN stack and input display e / enter / −−> process the user inout line b / base / tos −−> set the base to tos (1<tos>257) h / handled / −−> signal

					that an event handler handled the event and RPN should not handle the event
			T	Timer	tos −−> set the timer event to happen in tos/100 seconds
			z	zero	tos −−> the character used to diplay a given digit is given by c(d)=Z+d when Z!='0' and (d>9 ? 'A':'0')+d otherwise; RPN digits are case insensitive iff Z='0'=48; this byte code sets Z (so #'48'Uz is the normal mode)
T[t|m|h|D|M|Y]	time	−−> time	get time in ticks, minutes, hours, Days, Months, or Years		
D'text|button1|...'	dialog	−−> button	run dialog and report button pressed; example D'Do you like\RPN?|yes|no|maybe|' illustrates the current features of dialogs; the |buttons| are optional: D'Hello' is legal; if you start a button name with \ then it will be the default button which is selected if the user does something other than choose a button.		

?X	require	−−>	require X arguments to proceed; use '?@' to get X from the stack
S	sound	freq durAmp−−>	freq in Hz; durAmp=10*duration (in 100ths of a second) + amp[0−8]
KX	Key	−−>	simulate the user entering the character, X; if 'K' starts a function then a user Enter is not processed before running the function

Math

[Literal | Boolean | Control | Variables | Stack | **Math** | Trig | Misc]

Format	Name	Stack Description	Notes
+	add	nos tos −−> nos+tos	add
−	sub	nos tos −−> nos−tos	subtract
*	mult	nos tos −−> nos*tos	multiply
/	div	nos tos −−> nos/tos	divide
%	mod	nos tos −−> nos%tos	mod
P	power	nos tos −−> nos**tos	raise nos to the tos power
mXY	mul Int	tos −−> tos*XY	multiply by two digit integer XY
b	abs	tos −−> abs(tos)	absolute value
e	exp	tos −−> exp(tos)	e**tos
l	ln	tos −−> ln(tos)	natural log
L	log10	tos −−> log10(tos)	log base 10
n	negate	tos −−> −tos	negate
s	sqroot	tos −−> sqroot(tos)	square root
t	invert	tos −−> 1/tos	invert
f	fp	tos −−> fp(tos)	fractional part
w	wp	tos −−> wp(tos)	whole part
H	half	tos −−> tos/2	much faster than "2/"

Trig

[Literal | Boolean | Control | Variables | Stack | Math | **Trig** | Misc]

All trig operations are performed in the user's current trig mode (radians or degrees). Use the "Mr" (see Misc section) to test the current mode.

Format	Name	Stack Description	Notes
i	sin	tos −−> sin(tos)	sin
o	cos	tos −−> cos(tos)	cos
a	tan	tos −−> tan(tos)	tan
I	asin	tos −−> asin(tos)	asin
O	acos	tos −−> acos(tos)	acos
A	atan	tos −−> atan(tos)	atan

Stack

[Literal | Boolean | Control | Variables | **Stack** | Math | Trig | Misc]

RPN's stack codes all take a stack index an opcode. Indexes start at one (zero is not a valid stack index). Using an invalid stack index aborts the running function. The special opcode '@' (read 'd@' as "drop at") indicates that the stack index is on the stack. This stack index is an index to the stack after the index itself is removed from the stack (ie. d1 == 1d@).

Format	Name	Stack Description	Notes
gX	get	X+1 X...tos −−> X+1 X...tos X	copy the Xth stack item to the top; g1=dup; g2=over
pX	put	X+1 X X−1...tos −−> X+1 tos X−1...	put TOS into Xth slot on the stack; p1=drop; p2=swap drop
dX	drop	X+1 X...tos −−> X+1...tos	drop the Xth stack item; d1=drop
rX	rotate	X+1 X...tos −−> X+1...tos X	move Xth to top; r1=nop; r2=swap; r3=rot
kX	tuck	X+1 X...tos −−> X+1 tos X...nos	move top to Xth position; k1=nop; k2=swap; k3=tuck; tuck is the inverse of rotate (r3k3==nop if depth>=3)
h	depth	−−> depth	puts the stack depth on the stack (1 2 3 −−> 1 2 3 3)

Z	store	... −−> ...	store the whole stack temporarily (storage not persistent across relaunches of RPN and shared by all code); does not change the stack at all
z	recall	... −−> previous_stack	recall the last stored state of the stack (empty stack if nothing was stored)

Glossary

nos
: Next On Stack. The second value on the stack.

RPN−Code
: The language used to write RPN scripts.

stack gauge
: The vertical bar between the stack and script areas which indicates how full the stack is.

tos
: Top Of Stack. The first value on the stack.

Changes by Version

Version 3.61 −gt; 3.62 changes:

- Recordings now remember and replay the choices made in dialogs rather than present the dialog again.
- Fixed About... dialog display in Palm OS 3.x.
- Fixed key shortcuts for + and x^2.
- Improved accuracy of temperature conversions involving F degrees.
- Fixed Reinstall All... scripts menu action.
- Added 'UDx' bcode to queue up the choice for the next dialog.

Version 3.60 −> 3.61 changes:

- Fixed timer problem with About... Dialog.
- Fixed small drawing error.

Version 3.55 −> 3.60 changes:

- **Improved and expanded built−in scripts:**
  - **New Time Value of Money (TVM) functions.**
  - **New Conversion script with over 400 conversions.**
  - **New Time script with time arithmetic.**
- **Updated manual with script descriptions and new scripting information.**
- **Improved RPN scripting language:**
  - New RPN.4 format.
  - Valid subroutine names in RPN.4 code are any combination of printable, non−whitespace characters *not* in this list: "()'[]`{};
  - New RPN version bcode: Mv
  - Scripts can have upto 255 globals now, accessed with the x@ and X@ bcodes and defined in groups of 10 by capital letters in the header: RPN.4.B = 20 globals
  - Subroutines have precedence over locals now.
  - Removed single character access to globals.
  - Added single character subroutine calls by name.
  - Depricated .set and .make for local variables.
  - Added "=name" syntax for setting locals.
  - Local variable names restricted to [a−z|0−9|A−Z] and cannot start with a digit.
  - Depricated RPN.3 syntax because RPN.4 syntax is easier to use.
  - Return stack increased to 128 elements from 12. i.e. More recursion depth.
  - Fixed bug in using locals of same name in diffent scripts.
  - Changed the button for breaking into a running script from Enter to the Record button.
- Improved Script menu commands.
- Removed warning on tapping empty stack location.
- Changed Ok−Cancel confirmation dialogs to Yes−No dialogs.
- Script editor does not ask for Save confirmation unless title has changed.
- Removed Clock and Timer scripts (still available on the RPN Coweb).
- Export files can be imported, deleted, or preserved when RPN runs.

- Fixed removal of multiple scripts after importing at the same time.
- Renamed "Get Info..." menu "About..." and added some features to the about dialog.

Version 3.51 −> 3.55 changes:

- Fixed hotsync problem with older versions of Palm Desktop by making RPN multi−segment.
- Fixed bug in deleting scripts.
- Fixed incorrect link to bytecodes in manual.
- Fixed sin(90) error.

Version 3.50 −> 3.51 changes:

- **Updated interface and added tools for graphing.**
- Added 'New Script' command to editor menu.
- Fixed bug with ==, >=, <=, and != commands in scripts.
- Fixed export of databases from editor.
- Fixed initial insertion point focus in editor.
- Updated graphing documentation.

Version 3.22 −> 3.50 changes:

- **Built−in Script editing.**
- **Easy import and export of scripts.**
- Fixed color customization bug in OS 3.5, 8−bit devices.
- Beaming of scripts from script editor.
- Added ability to reset the key shortcuts and fixed potential endless loop after setting key shortcuts.
- tan(90) = numerical error now.
- Added end of script string, #end#.
- Key shortcuts can also be scripts defined in the "#keys#" script.
- Moved code entry into About dialog.
- New pricing structure, $29 professional, $19 student.
- Added word based commands to RPN−Code.
- Store (Z) and recall (z) stack bytecodes. Nop bytecode, `
- Access to globals as well as locals by name.
- This manual.

Version 3.21 −> 3.22 changes:

- Fixed graphing bug caused by compiler optimization.
- Fixed occasional bug in dragging, particularly on OS 3.1.
- Fixed problem with trace popup in some Sony devices (PEG−SJ30).
- Added Home button to graphing bounds form.

Version 3.20 −> 3.21 changes:

- Hi−Res icon for soft−graffiti display.
- Fixed dialog button selection problem.

Version 3.10 −> 3.20 changes:

- High Resolution support for main display.
- Customizable colors for main display.
- Dialog for user defined shortcut keys. Simplifying previous system.
- Fixed sound bug in dialogs.
- Fixed bug loading scripts with duplicate label ids.

Version 3.05 −> 3.10 changes:

- Palm OS 5 native ARM acceleration.
- Increased stack size to 80.
- Fixed bug blocking Tungsten center button action.
- Fixed bug in stack dragging.
- High−Res icons added.