RPLsh: An Interactive Shell for Stack-based Numerical Computation

Kevin P. Rauch

Dept. of Astronomy, University of Maryland, College Park, MD 20742

Abstract.

RPL shell[1], or RPLsh, is an interactive numerical shell designed to combine the convenience of a hand-held calculator with the computational power and advanced numerical functionality of a workstation. The user interface is modeled after stack-based scientific calculators such as those made by Hewlett-Packard (RPL[2] is the name of the Forth-like programming language used in the HP 48 series), but includes many features not found in hand-held devices, such as a multi-threaded kernel with job control, integrated extended precision arithmetic, a large library of special functions, and a dynamic, resizable window display. As a native C/C++ application, it is over 1000 times faster than HP 48 emulators (e.g., Emu48[3]) in simple benchmarks; for extended precision numerical analysis, its performance can exceed that of *Mathematica*® by similar amounts. Current development focuses on interactive user functionality, with comprehensive programming/debugging support to follow.

1. Introduction

The notion of progress in computer hardware has always been closely tied to the speed and sophistication with which numerical calculations can be performed. While the continuation of Moore's Law has resulted in hand-held devices more powerful than the 30 ton ENIAC machine, there is a growing gap between the speed and capabilities of hand-held scientific calculators as compared to desktop workstations—the result of both market forces (pricing pressure) and physical limitations (keyboard/screen size). Numerical analysis packages such as *Mathematica* or IDL, by contrast, are oriented towards script or command line processing, and lack the interactive amenities found in a traditional calculator.

RPL shell, or RPLsh, is designed to combine the convenience of a hand-held with the power of a desktop. Computation in RPLsh is based on a last-in, first-out operand stack, as for an RPN (Reverse Polish Notation) calculator or a PostScript interpreter; the stack may be of any depth and can hold objects of many different types, numeric and non-numeric. The RPLsh command

[1] `http://www.astro.umd.edu/~rauch/rplsh/`

[2] `http://www.hpmuseum.org/rpl.htm`

[3] `http://www.hpcalc.org/hp48/pc/emulators/`

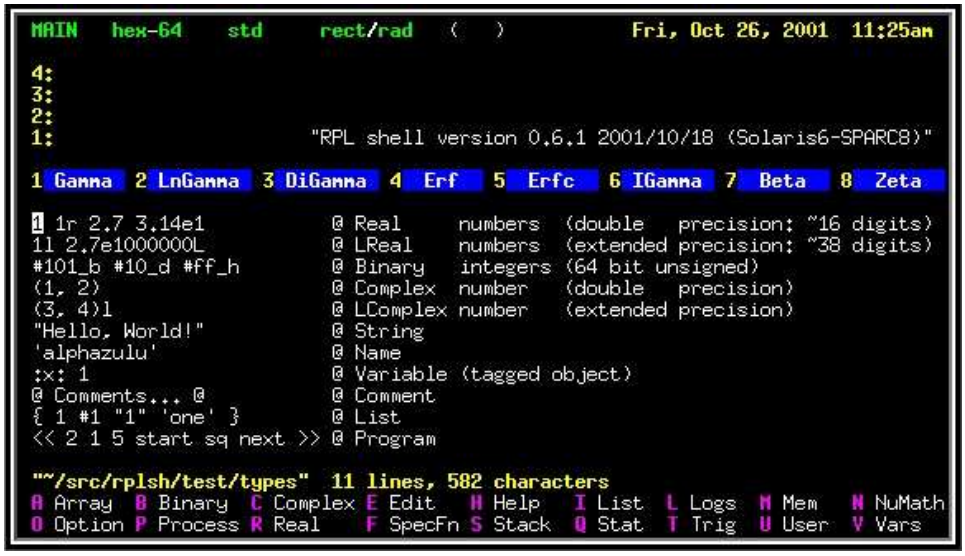

Figure 1. The RPLsh interactive display.

language is based on RPL ("Reverse Polish Lisp"), the procedural language invented by Hewlett-Packard and used in the HP 48 series; the look and feel of the RPLsh user interface is also similar to an HP 48. RPLsh is not, however, an HP 48 emulator—though a high degree of compatibility is an important design criterion. Nor is it intended to be a replacement for *Mathematica* or similar systems: its computational focus is numerics, not plotting or symbolic manipulation, which are well-served by a line-oriented input model.

Like traditional UNIX shells, RPL shell can operate either interactively, or non-interactively as an RPL script interpreter. Development currently centers on interactive features, to which the next section is devoted. Section 3 summarizes the object types currently implemented by RPLsh. Finally, future development plans are discussed in section 4.

2. User Interface

2.1. Display Layout

The graphical user display, shown in Figure 1, consists of several distinct sections; from top to bottom, these are: the status line, stack display, menu line, editor display, and menu quick reference. The status line summarizes the current machine state, such as the default base for binary input, the floating point output format, and the angle mode for trigonometric functions. Parentheses in the center of the status line indicate the state of the process queue; a single pair implies the machine is idle, a double pair means there are jobs in the queue, and a central `*` means a job is actively running. The stack display shows the formatted contents of the operand stack. The menu line enumerates the commands performed by function hot keys; to execute action n, press `ESC-`n or `F`n. User input occurs in the editor display. Finally, the menu references indicate

Table 1. Input Editor Active Keys

Key	Function	Key	Function
CTL-\	Quit RPLsh (panic exit).	CTL-Q	Execute the FgStart command.
CTL-A	Change input mode.	CTL-R	Redraw the display.
CTL-B	Activate stack browser.	CTL-S	Execute the FgStop command.
CTL-C	Cancel current action.	CTL-T	Insert --> ('To') into buffer.
CTL-D	Exit RPLsh.	CTL-U	Undo last executed command.
CTL-E	Execute the Eval command.	CTL-V	Visit/Edit level 1 stack object.
CTL-H	Backspace key.	CTL-W	Execute the Swap command.
CTL-I	Execute the Inv command.	CTL-X	Execute the Neg command.
CTL-J	Enter key.	CTL-Y	Activate command history display.
CTL-L	Execute the Last command.	CTL-Z	Suspend RPLsh, return to prompt.
CTL-M	Enter key.	[+-*/^]	Execute the arithmetic operation.
CTL-N	Display next menu line.	;	Push current buffer; create a new one.
CTL-P	Display previous menu line.	~	Begin an extended editor command.

the ESC-prefixed letter to use to switch to the given menu; e.g., pressing ESC-T (or ESC-t) switches the menu line to the Trig (trigonometric functions) menu. (Note: a number of the menus shown are currently empty and inaccessible.)

The window as a whole can be freely resized (12 rows by 40 columns minimum), and the available rows can be allocated as desired between the stack and editor displays via special editor commands.

2.2. Input Editor

RPLsh includes a small built-in editor suitable for most input needs; an external editor can also be used (cf. Table 2). A key feature of the built-in editor is that numerous keys are *active*, meaning that they execute specific RPLsh actions as soon as they are pressed; these are summarized in Table 1. The behavior of active keys also depends on the input mode, toggled via CTL-A. Extended commands, listed in Table 2, perform specialized operations. Use the normal cursor keys to move around the buffer, and the Insert key to toggle input between insert and replace modes. Parsing and execution of the input buffer occurs when you press Enter (or Return).

Table 2. Extended Input Editor Commands

Command[a]	Function
~i	Display RPLsh version information on the editor message line.
~e *nlines*	Set the number of displayed editor lines to *nlines*.
~s *nlines*	Set the number of displayed stack lines to *nlines*.
~r *filename*	Read file *filename* into the buffer (at the cursor).
~w *filename*	Write the (entire) editor buffer to file *filename*.
~v	Transfer buffer to a full-screen visual editor and reread on exit.

[a]Uppercase letters may also be used.

2.3. Interactive Environments

A stack browser allows the user to interactively view and manipulate specific stack objects, including swapping, duplicating, or deleting them. It is invoked with `CTL-B`; use the arrow keys to select an object, and press `q` or `CTL-C` to exit. A similar environment, called the autobrowser, is entered automatically whenever the input buffer is empty (and the stack is idle), and exited automatically when editor input occurs.

A history list of previous commands can be displayed by pressing `CTL-Y`. To edit and/or execute a previous input line, highlight it using the arrow keys and press `Enter`; exit the display by pressing `q` or `CTL-C`.

The process queue can be viewed via the Ps command, located in the Process menu (note that interactive environments *cannot* be invoked by typing the name; you must press the corresponding active key to enter). As the multithreaded kernel architecture of RPLsh (for Unix) allows the user to enter new tasks while others are still executing, the queue can, in principle, be quite large. The Ps display is dynamic and will update automatically as its contents change; one can also select specific jobs to pause or restart—but keep in mind that all (foreground) jobs share the same operand stack! Also bear in mind that when `stream=false` appears in the Fg (i.e., foreground) process entry, new jobs you create will *not* be executed until the queue is restarted (via the FgStart command, `CTL-Q`). As usual, press `q` or `CTL-C` to exit the Ps display.

3. Object Types

All object types implemented as of this writing (RPL shell version 0.6.1) are shown in the editor display of Figure 1. Note in particular that floating point numbers can be given a precision suffix; the default precision can be set with the Real and LReal commands. The default binary integer base operates similarly. Operations between numeric types will automatically promote both arguments to a common type, as appropriate; e.g., multiplying an LReal by a Complex will produce an LComplex. Also note that although a number of non-numeric types can be parsed as input, at the time of writing most of them have little to no associated functionality.

4. Future Development

Development of the interactive user interface has largely been completed; notable environments remaining to be implemented include a run-time options editor and the on-line help system. A number of important object types, including matrices and algebraic expressions, also remain unimplemented. As these basic types are introduced into the kernel, focus will shift towards object functionality—especially comprehensive programming support, which is currently quite limited. At this point, script-based, non-interactive use of RPL shell should be considered experimental; for everyday interactive use, however, RPLsh is already a stable and convenient numerical tool.