RCBScript — Libraries Format, Standard Libraries & vi-like Editor

This document contains work distribution for two people along with the technical details.

---

1) Library file format (proposal)

File extension:

- Source libraries: `*.rcblib`
- Packaged library archive: `*.rcblibpkg`

Manifest fields:

```
---
name: math
version: 0.1.0
rcb_version: 0.1
description: Basic math utilities for RCBScript
exports:
  - sum: (number, number) -> number
  - mul: (number, number) -> number
dependencies:
```

```
  - core: ">=0.1"
author: "Vineeth"
license: MIT
---
```

Modules section:

```
# modules
module math
func sum(a:number, b:number) -> number
    reply a + b

func mul(a:number, b:number) -> number
    reply a * b
endmodule
```

Loader responsibilities: validate manifest, type-check exports, register symbols, support imports.

---

2) Standard library layout

stdlib.rcblibpkg/

│

├── manifest.yaml

├── io.rcblib

├── math.rcblib

└── strings.rcblib

- Naming rules: lowercase + underscore.
- Symbol resolution: `import math`, `from math import sum`, aliasing supported.

---

3) Starter library examples

`Io.rcblib`

```
manifest:
  name: io
  version: 1.0
  description: "Basic input/output utilities for RCBScript"
  exports:
    - print
    - input

module io
```

```
  func print(msg:text) -> void
      results(msg)


  func input(prompt:text) -> text
      results(prompt)
      reply "user_entered_value"
```

Math.rcblib

```
manifest:
  name: math
  version: 1.0
  description: "Mathematical utilities (sum, pow, sqrt)"
  exports:
    - sum
    - pow
    - sqrt


module math
  func sum(a:number, b:number) -> number
      reply a + b
```

```
func pow(base:number, exp:number) -> number
    result:number = 1
    loop from i:number = 1 to exp
        result = result * base
    reply result

func sqrt(x:number) -> number
    guess:number = x / 2
    loop while abs(guess*guess - x) > 0.0001
        guess = (guess + x/guess) / 2
    reply guess
```

Strings.rcblib

```
manifest:
  name: strings
  version: 1.0
  description: "String utilities (concat, length, toUpper, toLower)"
  exports:
    - concat
```

```
    - length
    - toUpper
    - toLower


module strings
  func concat(a:text, b:text) -> text
      reply a + b


  func length(s:text) -> number
      reply len(s)


  func toUpper(s:text) -> text
      reply "TO_UPPER(" + s + ")"


  func toLower(s:text) -> text
      reply "TO_LOWER(" + s + ")"
```

---

4) Package Manifest (`manifest.yaml`)

```
package:
```

```
name: stdlib
version: 1.0
description: "RCBScript Standard Library (I/O, Math, Strings)"
modules:
  - io
  - math
  - strings
```

---

5) Testing integration

Example `math_tests.rcbtest`:

```
import math

test "sum basic"
expect math.sum(2,3) is 5
```

---

6) Plan for creating libraries

1. Manifest parser (YAML-lite)
2. Library registry (name → symbols)
3. `import` handling in compiler frontend
4. Create stdlib package with I/O, math, and strings
5. Wire runtime to load `.rcblib` files and `.rcblibpkg` archives

---

7) Simple vi-like editor (plan)

- Modes: COMMAND and INSERT
- Navigation: h, j, k, l
- Commands: `:w`, `:q`, `:wq`, `:e filename`
- Status bar with filename, mode, position
- Implemented in Python with `curses`

---

8) Starter editor code (`vi_clone.py`)

Prototype implemented in Python using curses (supports insert mode, navigation, basic commands).

```
#!/usr/bin/env python3
# vi_clone.py -- minimal vi-like editor prototype (MVP)
# Usage: python3 vi_clone.py [file.rcb]
import curses
import sys
```

```
class Buffer:
    def __init__(self, lines=None):
        self.lines = lines or [""]
        self.cx = 0
        self.cy = 0

    def insert(self, ch):
        line = self.lines[self.cy]
        self.lines[self.cy] = line[:self.cx] + ch + line[self.cx:]
        self.cx += len(ch)

    def delete_back(self):
        if self.cx == 0:
            if self.cy == 0:
                return
            prev = self.lines[self.cy-1]
            self.cx = len(prev)
            self.lines[self.cy-1] = prev + self.lines[self.cy]
            del self.lines[self.cy]
            self.cy -= 1
        else:
            line = self.lines[self.cy]
            self.lines[self.cy] = line[:self.cx-1] + line[self.cx:]
            self.cx -= 1

    def newline(self):
        line = self.lines[self.cy]
        left = line[:self.cx]
        right = line[self.cx:]
        self.lines[self.cy] = left
        self.lines.insert(self.cy+1, right)
        self.cy += 1
        self.cx = 0

    def move_left(self):
        if self.cx > 0:
            self.cx -= 1
        elif self.cy > 0:
            self.cy -= 1
```

```
            self.cx = len(self.lines[self.cy])

    def move_right(self):
        if self.cx < len(self.lines[self.cy]):
            self.cx += 1
        elif self.cy < len(self.lines)-1:
            self.cy += 1
            self.cx = 0

    def move_up(self):
        if self.cy > 0:
            self.cy -= 1
            self.cx = min(self.cx, len(self.lines[self.cy]))

    def move_down(self):
        if self.cy < len(self.lines)-1:
            self.cy += 1
            self.cx = min(self.cx, len(self.lines[self.cy]))

def editor(stdscr, filename=None):
    curses.raw()
    curses.noecho()
    stdscr.keypad(True)
    maxy, maxx = stdscr.getmaxyx()

    if filename:
        try:
            with open(filename, 'r', encoding='utf-8') as f:
                lines = [l.rstrip('\\n') for l in f.readlines()]
        except FileNotFoundError:
            lines = [""]
    else:
        lines = [""]

    buf = Buffer(lines)
    mode = 'COMMAND'
    cmd = ''

    # screen offset for scrolling
```

```
    top = 0
    left = 0

    while True:
        stdscr.clear()
        maxy, maxx = stdscr.getmaxyx()
        # render text with simple horizontal and vertical clipping
        height = maxy - 1
        for i in range(height):
            line_idx = top + i
            if line_idx >= len(buf.lines):
                break
            line = buf.lines[line_idx]
            visible = line[left:left+maxx-1]
            try:
                stdscr.addstr(i, 0, visible)
            except curses.error:
                pass  # ignore drawing errors for lines that barely fit

        # status bar
        fname = filename or '[no file]'
        status = f\"{fname} -- {mode} -- {buf.cy+1}:{buf.cx+1}  \"
        # show a small message for command input
        if cmd:
            status += f\":{cmd}\"
        status = status[:maxx-1]
        try:
            stdscr.addstr(maxy-1, 0, status)
        except curses.error:
            pass

        # cursor position on screen
        screen_y = buf.cy - top
        screen_x = buf.cx - left
        # clamp
        if screen_y < 0:
            top = buf.cy
            screen_y = 0
        elif screen_y >= height:
```

```
            top = buf.cy - height + 1
            screen_y = buf.cy - top

        if screen_x < 0:
            left = buf.cx
            screen_x = 0
        elif screen_x >= maxx-1:
            left = buf.cx - (maxx-2)
            screen_x = buf.cx - left

        try:
            stdscr.move(screen_y, screen_x)
        except curses.error:
            pass
        stdscr.refresh()

        try:
            c = stdscr.get_wch()
        except KeyboardInterrupt:
            return

        if mode == 'COMMAND':
            if isinstance(c, str):
                if c == 'i':
                    mode = 'INSERT'
                elif c == 'h':
                    buf.move_left()
                elif c == 'j':
                    buf.move_down()
                elif c == 'k':
                    buf.move_up()
                elif c == 'l':
                    buf.move_right()
                elif c == ':':
                    # enter command-line mode
                    cmd = ''
                    # small command input loop
                    while True:
                        ch = stdscr.get_wch()
```

```
                        if ch == '\n' or ch == '\r':
                            break
                        elif ch == '\x1b':  # ESC to cancel
                            cmd = ''
                            break
                        elif isinstance(ch, str):
                            cmd += ch
                            # redraw status with command
                            try:
                                stdscr.addstr(maxy-1, 0, (f\":\" +
cmd)[:maxx-1])
                                stdscr.clrtoeol()
                            except curses.error:
                                pass
                            stdscr.refresh()
                    # process command
                    if cmd == 'q':
                        return
                    elif cmd == 'w':
                        if filename:
                            with open(filename, 'w', encoding='utf-8')
as f:
                                f.write('\\n'.join(buf.lines) + '\\n')
                        cmd = ''
                    elif cmd == 'wq':
                        if filename:
                            with open(filename, 'w', encoding='utf-8')
as f:
                                f.write('\\n'.join(buf.lines) + '\\n')
                        return
                    elif cmd.startswith('e '):
                        _, newf = cmd.split(' ', 1)
                        filename = newf.strip()
                        try:
                            with open(filename, 'r', encoding='utf-8')
as f:
                                buf.lines = [l.rstrip('\\n') for l in
f.readlines()]
                        except FileNotFoundError:
```

```
                            buf.lines = ['']
                        buf.cx = buf.cy = 0
                    cmd = ''
                elif c == 'G':
                    # go to end of file
                    buf.cy = len(buf.lines) - 1
                    buf.cx = 0
                elif c == '0':
                    # go to start of line
                    buf.cx = 0
                elif c == '$':
                    buf.cx = len(buf.lines[buf.cy])
                # allow arrow keys too
            elif c == curses.KEY_LEFT:
                buf.move_left()
            elif c == curses.KEY_RIGHT:
                buf.move_right()
            elif c == curses.KEY_UP:
                buf.move_up()
            elif c == curses.KEY_DOWN:
                buf.move_down()
        else:  # INSERT mode
            if isinstance(c, str):
                if c == '\x1b':  # ESC
                    mode = 'COMMAND'
                elif c == '\n':
                    buf.newline()
                elif c == '\x7f':  # backspace (sometimes)
                    buf.delete_back()
                else:
                    buf.insert(c)
            else:
                if c == curses.KEY_BACKSPACE:
                    buf.delete_back()

if __name__ == '__main__':
    filename = sys.argv[1] if len(sys.argv) > 1 else None
    try:
        curses.wrapper(editor, filename)
```

```
    except Exception as e:
        print('Editor exited:', e)
```

---

9) Work Distribution

Vineeth

- Define and finalize **library file format** (manifest, modules, packaging rules).
- Build standard libraries (`io.rcblib`, `math.rcblib`, `strings.rcblib`).
- Create test cases for libraries (`math_tests.rcbtest`).
- Implement manifest parser and library registry (prototype in Python).
- Deliverables:
    - `io.rcblib`, `math.rcblib`, `strings.rcblib`
    - Parser + registry prototype
    - Example tests

Nayak

- Plan and implement **vi-like editor** (prototype).
- Implement modes (command/insert), navigation, and basic commands.
- Add file saving/loading functionality.
- Extend later with scrolling, syntax highlighting.
- Deliverables:

- `vi_clone.py` (working prototype)
- Documentation of features & usage
- Roadmap for advanced features (search, undo, highlighting)
