object-browser-strategy

August 16, 2025 at 10:07 AM

zzvim-R Object Browser Strategy

Research Analysis: Current Object Browsing Approaches

Based on research of RStudio and R.nvim object browsing strategies, this document outlines a recommended approach for adding object browsing to zzvim-R that aligns with its core philosophy.

Current Object Browsing Strategies

RStudio Approach

- **Visual Environment Panel**: GUI-based workspace browser with hierarchical display
- **Real-time Updates**: Automatic refresh when objects are created/destroyed
- **Interactive Features**: Click-to-inspect, filtering, sorting by object type/size
- **Data Viewer**: Spreadsheet-like interface for data.frame inspection
- **Memory Usage**: Shows object sizes and memory consumption

R.nvim Approach

- **Dedicated Buffer Window**: Opens object browser in separate Neovim buffer
- **Text-based Hierarchy**: Tree-like display with expandable data.frame columns
- **Color Coding**: Different colors for object types (functions, data, lists)
- **Live Updates**: Real-time synchronization with R session via nvimcom
- **Buffer Integration**: Objects clickable for immediate inspection

Proposed zzvim-R Object Browser Strategy

Following zzvim-R's philosophy of **terminal-based simplicity** with **smart R integration**, here's the recommended approach:

Core Philosophy Alignment

- **Terminal-Native**: Use terminal buffers, not GUI elements
- **Command-Driven**: R functions + Vim text manipulation
- **Lightweight**: Minimal memory overhead, no external dependencies
- **Pattern-Based**: Leverage existing R commands with intelligent formatting

Implementation Approach: "Smart Terminal Browser"

```
" Core object browser implementation
function! s:RBrowseWorkspace() abort
    " Create or switch to browser buffer
    let browser_buf = s:GetOrCreateBrowserBuffer()

    " Generate R workspace listing command
    let r_cmd = s:GenerateWorkspaceCommand()

    " Execute and capture output
    call s:UpdateBrowserContent(r_cmd)

    " Apply syntax highlighting and key mappings
    call s:SetupBrowserBuffer()
endfunction

function! s:GenerateWorkspaceCommand() abort
    " Use R's built-in functions with formatting
    return join([
        'cat("=== R Workspace Browser ===\\n")',
        'ls_output <- capture.output({',
        '  objs <- ls(.GlobalEnv)',
        '  if(length(objs) > 0) {',
        '    for(obj in objs) {',
        '      obj_class <- class(get(obj))[1]',
        '      obj_size <- format(object.size(get(obj)), units="auto")',
        '      obj_dim <- if(is.data.frame(get(obj)) || is.matrix(get(obj))) {',
        '        paste0(" [", paste(dim(get(obj)), collapse="x"), "]")',
        '      } else if(is.vector(get(obj))) {',
        '        paste0(" [", length(get(obj)), "]")',
        '      } else ""',
        '      cat(sprintf("%-20s %s%s (%s)\\\\n", obj, obj_class, obj_dim, obj_size))'
        '    }',
        '  } else {',
        '    cat("No objects in workspace\\\\n")',
        '  }',
        '})',
        'cat(ls_output, sep="\\\\n")'
    ], '; ')
endfunction
```

Feature Set: Progressive Enhancement

Level 1: Basic Workspace Listing

```
" Quick workspace overview
command! RWorkspace call s:RBrowseWorkspace()
nnoremap <LocalLeader>rw :RWorkspace<CR>

" Show in scratch buffer with smart formatting:
" === R Workspace Browser ===
" data_clean          data.frame [1000x12] (1.2 Mb)
" model_results       list [4] (15.6 Kb)
" plot_theme          ggplot [1] (2.3 Kb)
" custom_function     function [1] (856 bytes)
```

Level 2: Interactive Object Inspection

```
" Browser buffer key mappings
function! s:SetupBrowserBuffer() abort
    " Object inspection on current line
    nnoremap <buffer> <CR> :call <SID>InspectObjectUnderCursor()<CR>
    nnoremap <buffer> s :call <SID>StrObjectUnderCursor()<CR>
    nnoremap <buffer> h :call <SID>HeadObjectUnderCursor()<CR>
    nnoremap <buffer> d :call <SID>DimObjectUnderCursor()<CR>

    " Refresh workspace
    nnoremap <buffer> r :call <SID>RefreshWorkspace()<CR>

    " Close browser
    nnoremap <buffer> q :close<CR>
endfunction

function! s:InspectObjectUnderCursor() abort
```

```
    let obj_name = s:ExtractObjectName()
    if !empty(obj_name)
        call s:RAction('str', obj_name)
    endif
endfunction
```

Level 3: Data Frame Column Browser

```
" Expandable data.frame inspection
function! s:ExpandDataFrame() abort
    let df_name = s:ExtractObjectName()
    let expand_cmd = printf('cat("Columns in %s:\\\\n"); str(%s)', df_name, df_name)
    call s:Send_to_r(expand_cmd, 1)
endfunction

" Browser display with expandable sections:
" data_clean          data.frame [1000x12] (1.2 Mb) [+]
"   id                numeric [1000] (8.0 Kb)
"   name              character [1000] (24.5 Kb)
"   value             numeric [1000] (8.0 Kb)
```

Level 4: Smart Filtering and Search

```
" Filter workspace by pattern
command! -nargs=1 RWorkspaceFilter call s:FilterWorkspace(<q-args>)

function! s:FilterWorkspace(pattern) abort
    let filter_cmd = printf('cat("Objects matching \'%s\':\\\\n"); ls(pattern="%s")',
                           \\ a:pattern, a:pattern)
    call s:UpdateBrowserContent(filter_cmd)
endfunction

" Quick filters
nnoremap <LocalLeader>rwd :RWorkspaceFilter data.frame<CR>
nnoremap <LocalLeader>rwf :RWorkspaceFilter function<CR>
nnoremap <LocalLeader>rwl :RWorkspaceFilter list<CR>
```

Integration with Existing zzvim-R Features

Buffer-Specific Workspaces

```
" Integrate with multi-terminal architecture
function! s:GetWorkspaceForCurrentBuffer() abort
    let terminal_name = s:GetTerminalName()
    let r_cmd = printf('if(exists(".zzvim_buffer_id")) cat("Buffer: %s\\\\n")',
                     \\ expand('%:t'))
    return r_cmd . '; ' . s:GenerateWorkspaceCommand()
endfunction
```

Smart Object Detection

```
" Leverage existing pattern recognition
function! s:ShowObjectsInCurrentFunction() abort
    let func_lines = s:GetCodeBlock()  " Reuse existing function
    " Parse function for object assignments and show relevant workspace subset
endfunction
```

Terminal-First Design Benefits

1. **Consistency**: Uses same terminal communication as existing features
2. **Performance**: R-native commands, no external parsing needed

3. **Flexibility**: Easy to extend with additional R functions
4. **Reliability**: Leverages proven temp file approach for complex commands
5. **Integration**: Works seamlessly with buffer-specific terminals

Progressive Implementation Plan

Phase 1 (Immediate - 1 week):

```
" Basic workspace listing in scratch buffer
command! RWorkspace call s:RWorkspace()
function! s:RWorkspace() abort
    let cmd = 'cat("=== Workspace ===\\\\n"); print(ls.str())'
    call s:Send_to_r(cmd, 1)
endfunction
```

Phase 2 (1-2 weeks):

- Dedicated browser buffer with object inspection
- Key mappings for immediate object analysis
- Integration with existing `<LocalLeader>` shortcuts

Phase 3 (2-4 weeks):

- Smart formatting and object size display

- Data frame column expansion
- Filtering and search capabilities

Phase 4 (Future):

- Visual indicators for object changes
- History of workspace modifications
- Export workspace summaries

Example Browser Output

```
=== R Workspace Browser ===
Buffer: analysis.R

data_raw            data.frame [5000x15] (2.1 Mb)
data_clean          data.frame [4800x12] (1.8 Mb)
model_lm            lm [12] (45.2 Kb)
model_glm           glm [30] (67.8 Kb)
plot_theme          theme [50] (12.3 Kb)
custom_transform    function [1] (1.2 Kb)
temp_results        list [8] (234.5 Kb)

Key mappings:
<CR>  - Inspect object (str)
s     - Structure (str)
h     - Head preview
d     - Dimensions
r     - Refresh workspace
q     - Close browser
```

Advantages Over Competitors

vs. RStudio: - **Lightweight**: No GUI overhead, terminal-native - **Keyboard-driven**: No mouse required, Vim-style navigation - **Scriptable**: Pure R commands, easily customizable

vs. R.nvim: - **Simpler**: No external dependencies or complex protocols - **Terminal-based**: Consistent with zzvim-R architecture

- **R-native**: Uses standard R functions, no custom parsing

Alignment with zzvim-R Philosophy

[CHECKMARK] Core Principles Maintained: - **Terminal-based communication**: Uses existing `s:Send_to_r()` infrastructure - **Smart R pattern integration**: Leverages R's built-in workspace functions - **Minimal external dependencies**: Pure VimScript + R, no additional packages - **Educational VimScript examples**: Clear, documented implementation patterns - **Buffer-specific isolation**: Integrates with multi-terminal architecture - **Silent execution with immediate feedback**: Consistent user experience

Implementation Priority

This object browser feature represents a **high-impact, medium-effort** enhancement that:

1. **Addresses user pain points** identified in competitive analysis
2. **Maintains architectural consistency** with existing zzvim-R design
3. **Provides incremental value** through progressive implementation phases
4. **Enhances competitive positioning** without compromising core philosophy

The proposed approach offers modern workspace management capabilities while preserving zzvim-R's fundamental strengths: simplicity, reliability, and terminal-native operation.

Next Steps

1. **Implement Phase 1**: Basic `RWorkspace` command for immediate user value
2. **User Testing**: Gather feedback on basic functionality and workflow integration

3. **Iterative Enhancement**: Progress through phases based on user needs and feedback
4. **Documentation**: Integrate with existing help system and user guides
5. **Performance Optimization**: Ensure minimal impact on existing zzvim-R performance