

# Documentation Bundle

- **Root:** C:\Users\dacoo\OneDrive\Topos\_ofHeaven\C700Heaven\Eternity
- **Generated:** 2026-02-06T01:22:02
- **Included files:** 111
- **Max text file bytes:** 2000000
- **Ignored dirs:** .git, .hg, .idea, .svn, .venv, .vscode, \_\_pycache\_\_, build, dist, doc, node\_modules, vendor, venv

## Filesystem Tree (included paths)

```
Eternity
├── Architecture
│   ├── A-0_System.txt
│   ├── A_plus.txt
│   ├── A_plus_plus.txt
│   ├── A_sharp_axiom.txt
│   ├── A_sharp_dot-NET.txt
│   ├── assembly.txt
│   ├── C.txt
│   ├── vector.png
│   └── vector.txt
├── Art
│   ├── Stone_12.png
│   └── Stone_39.png
└── GenesisOS
    ├── apps.php
    ├── auth.php
    ├── bootstrap.php
    ├── charset-grid-viewer.js
    ├── charset-grid-viewer.json
    ├── clix-repl.html
    ├── clix.js
    ├── clix.json
    ├── config.php
    ├── databank-manager.js
    ├── databank-manager.json
    ├── db-json-repl.js
    ├── db-json-repl.json
    ├── digital-linguist.js
    ├── digital-linguist.json
    ├── docs.py
    ├── filesystem.php
    ├── fsm_pipeline.cpp
    ├── improvements_to_be_made.md
    ├── index.css
    ├── index.php
    ├── lattice-gamev4.js
    └── lattice-gamev4.json
```

```
    └── make_docs_pdf.py
    └── memory.txt
    └── quadtree-visualizer.js
    └── quadtree-visualizer.json
    └── repl-computer.js
    └── repl-computer.json
    └── sandbox.html
    └── sandbox.php
    └── security_bootstrap.php
    └── solution.cpp
    └── system.php
    └── text_reversion.txt
    └── tr-api.php
    └── tr-auth.php
    └── tr-generate_hash.php
    └── tr-index.php
    └── tr-login.php
    └── tr-programming.js
    └── tr-script.js
    └── tr-style.css
    └── tr-terminal-config.json
    └── tr-terminal.js
    └── tr-terminal.json
    └── tr-VersionManager.php
    └── users.php
    └── x000000000.txt
    └── x000000001.txt
└── Hardware
    ├── out_cat4
    │   ├── category.json
    │   ├── category.png
    │   ├── category_assembly.png
    │   ├── category_manifest.json
    │   └── grid.png
    ├── outnn_4_30-17
    │   ├── assembly.png
    │   ├── classical.png
    │   ├── quantum.png
    │   ├── source_classical.txt
    │   └── source_quantum.txt
    ├── outny_4_30-17
    │   ├── assembly.png
    │   ├── classical.png
    │   ├── quantum.png
    │   ├── source_classical.txt
    │   └── source_quantum.txt
    ├── outyn_4_30-17
    │   ├── assembly.png
    │   ├── classical.png
    │   ├── quantum.png
    │   ├── source_classical.txt
    │   └── source_quantum.txt
    └── outyy_4_30-17
        └── assembly.png
```

```
    └── classical.png
    └── quantum.png
    └── source_classical.txt
        └── source_quantum.txt
    └── C700h-stable.py
    └── C700h_updated.md
    └── C700h_updated.py
    └── integerDatabase
        ├── operator-alphabet.js
        ├── operator-build.js
        ├── operator-dropdown.js
        ├── operator-extensions.js
        ├── operator-framework.js
        ├── operator-language.js
        ├── operator-map.js
        ├── operator-persistence.js
        ├── operator.css
        ├── operator.html
        └── operator.js
    └── Linecraft_cpp_with_emit_rootfix2
        ├── files
        │   ├── x00001.txt
        │   └── x00002.txt
        ├── plugins
        │   ├── echo
        │   │   ├── echo_plugin.py
        │   │   └── plugin.json
        │   └── emit
        │       ├── emit_plugin.py
        │       └── plugin.json
        ├── src
        │   └── linecraft_cli.cpp
        ├── CMakeLists.txt
        ├── CMakePresets.json
        └── README.md
    └── bundle_documentation.py
```

## Table of Contents

1. [Architecture/A-0\\_System.txt](#)
2. [Architecture/A\\_plus.txt](#)
3. [Architecture/A\\_plus\\_plus.txt](#)
4. [Architecture/A\\_sharp\\_axiom.txt](#)
5. [Architecture/A\\_sharp\\_dot-NET.txt](#)
6. [Architecture/assembly.txt](#)
7. [Architecture/C.txt](#)
8. [Architecture/vector.png](#)
9. [Architecture/vector.txt](#)
10. [Art/Stone\\_12.png](#)
11. [Art/Stone\\_39.png](#)
12. [bundle\\_documentation.py](#)

13. [GenesisOS/apps.php](#)
14. [GenesisOS/auth.php](#)
15. [GenesisOS/bootstrap.php](#)
16. [GenesisOS/charset-grid-viewer.js](#)
17. [GenesisOS/charset-grid-viewer.json](#)
18. [GenesisOS/clix-repl.html](#)
19. [GenesisOS/clix.js](#)
20. [GenesisOS/clix.json](#)
21. [GenesisOS/config.php](#)
22. [GenesisOS/databank-manager.js](#)
23. [GenesisOS/databank-manager.json](#)
24. [GenesisOS/db-json-repl.js](#)
25. [GenesisOS/db-json-repl.json](#)
26. [GenesisOS/digital-linguist.js](#)
27. [GenesisOS/digital-linguist.json](#)
28. [GenesisOS/docs.py](#)
29. [GenesisOS/filesystem.php](#)
30. [GenesisOS/fsm\\_pipeline.cpp](#)
31. [GenesisOS/improvements\\_to\\_be\\_made.md](#)
32. [GenesisOS/index.css](#)
33. [GenesisOS/index.php](#)
34. [GenesisOS/lattice-gamev4.js](#)
35. [GenesisOS/lattice-gamev4.json](#)
36. [GenesisOS/make\\_docs\\_pdf.py](#)
37. [GenesisOS/memory.txt](#)
38. [GenesisOS/quadtrees-visualizer.js](#)
39. [GenesisOS/quadtrees-visualizer.json](#)
40. [GenesisOS/repl-computer.js](#)
41. [GenesisOS/repl-computer.json](#)
42. [GenesisOS/sandbox.html](#)
43. [GenesisOS/sandbox.php](#)
44. [GenesisOS/security\\_bootstrap.php](#)
45. [GenesisOS/solution.cpp](#)
46. [GenesisOS/system.php](#)
47. [GenesisOS/text\\_reversion.txt](#)
48. [GenesisOS/tr-api.php](#)
49. [GenesisOS/tr-auth.php](#)
50. [GenesisOS/tr-generate\\_hash.php](#)
51. [GenesisOS/tr-index.php](#)
52. [GenesisOS/tr-login.php](#)
53. [GenesisOS/tr-programming.js](#)
54. [GenesisOS/tr-script.js](#)
55. [GenesisOS/tr-style.css](#)
56. [GenesisOS/tr-terminal-config.json](#)
57. [GenesisOS/tr-terminal.js](#)
58. [GenesisOS/tr-terminal.json](#)

59. [GenesisOS/tr-VersionManager.php](#)
60. [GenesisOS/users.php](#)
61. [GenesisOS/x000000000.txt](#)
62. [GenesisOS/x000000001.txt](#)
63. [Hardware/C700h-stable.py](#)
64. [Hardware/C700h\\_updated.md](#)
65. [Hardware/C700h\\_updated.py](#)
66. [Hardware/out\\_cat4/category.json](#)
67. [Hardware/out\\_cat4/category.png](#)
68. [Hardware/out\\_cat4/category\\_assembly.png](#)
69. [Hardware/out\\_cat4/category\\_manifest.json](#)
70. [Hardware/out\\_cat4/grid.png](#)
71. [Hardware/outnn\\_4\\_30-17/assembly.png](#)
72. [Hardware/outnn\\_4\\_30-17/classical.png](#)
73. [Hardware/outnn\\_4\\_30-17/quantum.png](#)
74. [Hardware/outnn\\_4\\_30-17/source\\_classical.txt](#)
75. [Hardware/outnn\\_4\\_30-17/source\\_quantum.txt](#)
76. [Hardware/outny\\_4\\_30-17/assembly.png](#)
77. [Hardware/outny\\_4\\_30-17/classical.png](#)
78. [Hardware/outny\\_4\\_30-17/quantum.png](#)
79. [Hardware/outny\\_4\\_30-17/source\\_classical.txt](#)
80. [Hardware/outny\\_4\\_30-17/source\\_quantum.txt](#)
81. [Hardware/outyn\\_4\\_30-17/assembly.png](#)
82. [Hardware/outyn\\_4\\_30-17/classical.png](#)
83. [Hardware/outyn\\_4\\_30-17/quantum.png](#)
84. [Hardware/outyn\\_4\\_30-17/source\\_classical.txt](#)
85. [Hardware/outyn\\_4\\_30-17/source\\_quantum.txt](#)
86. [Hardware/outyy\\_4\\_30-17/assembly.png](#)
87. [Hardware/outyy\\_4\\_30-17/classical.png](#)
88. [Hardware/outyy\\_4\\_30-17/quantum.png](#)
89. [Hardware/outyy\\_4\\_30-17/source\\_classical.txt](#)
90. [Hardware/outyy\\_4\\_30-17/source\\_quantum.txt](#)
91. [integerDatabase/operator-alphabet.js](#)
92. [integerDatabase/operator-build.js](#)
93. [integerDatabase/operator-dropdown.js](#)
94. [integerDatabase/operator-extensions.js](#)
95. [integerDatabase/operator-framework.js](#)
96. [integerDatabase/operator-language.js](#)
97. [integerDatabase/operator-map.js](#)
98. [integerDatabase/operator-persistence.js](#)
99. [integerDatabase/operator.css](#)
100. [integerDatabase/operator.html](#)
101. [integerDatabase/operator.js](#)
102. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/CMakeLists.txt](#)
103. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/CMakePresets.json](#)
104. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/files/x00001.txt](#)

105. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/files/x00002.txt](#)
106. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/plugins/echo/echo\\_plugin.py](#)
107. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/plugins/echo/plugin.json](#)
108. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/plugins/emit/emit\\_plugin.py](#)
109. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/plugins/emit/plugin.json](#)
110. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/README.md](#)
111. [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/src/linecraft\\_cli.cpp](#)

## File Contents

### [1] [Architecture/A-0\\_System.txt](#)

- **Bytes:** 6069
- **Type:** text

```

; ======:contentReference[oaicite:4]{index=4}=====
; A-0 SYSTEM SCRIPT (monolithic): SYSTEM_SAFE_GENERATOR.A0
; =====
; Program intent (from attached assembly):
;   - Open "system_safe.txt" for writing
;   - Enumerate all base-64 strings of length N=4 using a 64-char alphabet
;   - Write each string + newline
;   - Close file, return status
;
; Alphabet (64 chars):
;   "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 \n"
; =====

; -----
; DATA DECK (literals)
; -----
D0001: ASCII "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 \n"
; 64 chars
D0002: ASCII "system_safe.txt"
D0003: ASCII "w"
D0004: ASCII "Error opening file"
D0005: U64    4 ; N = 4
D0006: U64    64 ; BASE = 64
D0007: U64    63 ; MASK =
0x3F
D0008: U8     10 ; '\n'

; -----
; RESERVED STORAGE (variables)
; -----
V0001: PTR    0      ; FILE_HANDLE
V0002: U64    0      ; LIMIT = 64^N
V0003: U64    0      ; I (outer counter)
V0004: U64    0      ; X (scratch copy of I)
V0005: S64    0      ; J (signed index, counts down)
V0006: U64    0      ; DIGIT (0..63)

```

```

V0007: U8      0          ; CH
V0010: BYTES 64           ; BUF (we only use first N bytes)

;
; -----
; SUBROUTINE DIRECTORY (numeric "call numbers")
; -----
; ID      NAME             ARGS...          -> RET
; -----
; 010    U64_SAFE_POW     (base, exp, out_u64_ptr)   -> ok(0/1)
; 020    GEN_PERMUTATIONS (n, alphabet_ptr, file_handle) -> ok(0/1)
; 500    FOPEN             (filename_ptr, mode_ptr)    ->
file_handle(0 if fail)
; 501    FCLOSE            (file_handle)        -> ok(0/1)
; 502    FWRITE_ALL         (file_handle, buf_ptr, len_u64) -> ok(0/1)
; 503    FPUTC_SAFE         (file_handle, u8_char)    -> ok(0/1)
; 510    UMUL_OVF          (a_u64, b_u64)       ->
(prod_u64, ovf_u8)
; 520    PERROR            (msg_ptr)          -> ()
; 999    EXIT              (code_u64)         -> ()
;
; (500..503,520,999 are "environment/library" routines in A-0 spirit.)

;
; =====
; SUBROUTINE 010: U64_SAFE_POW(base, exp, out_ptr) -> ok
; Mirrors: safe_uint64_power() in the assembly
; =====
SUB 010 U64_SAFE_POW
    ; out = 1
    STORE_U64 [ARG3], 1
    STORE_U64 V0003, 0           ; reuse V0003 as local COUNTER

L010_00:
    ; if COUNTER >= EXP: return 1
    IF_U64_GE V0003, ARG2 GOTO L010_OK

    ; (prod, ovf) = UMUL_OVF(*out, base)
    LOAD_U64 V0004, [ARG3]
    CALL 510 V0004, ARG1      -> V0002, V0007 ; prod -> V0002, ovf -> V0007

    ; if ovf != 0: return 0
    IF_U8_NE V0007, 0          GOTO L010_FAIL

    ; *out = prod
    STORE_U64 [ARG3], V0002

    ; COUNTER++
    ADD_U64 V0003, V0003, 1
    GOTO L010_00

L010_FAIL:
    RETURN_U8 0

L010_OK:
    RETURN_U8 1

```

```
ENDSUB
```

```
; =====
; SUBROUTINE 020: GEN_PERMUTATIONS(n, alphabet_ptr, file_handle) -> ok
; Mirrors: generate_permutations() in the assembly
; =====
SUB 020 GEN_PERMUTATIONS
    ; compute LIMIT = 64^n safely
    CALL 010 D0006, ARG1, &V0002      -> V0007      ; ok in V0007
    IF_U8_EQ V0007, 0                 GOTO L020_FAIL

    ; i = 0
    STORE_U64 V0003, 0

L020_I_LOOP:
    ; if i >= LIMIT: return 1
    IF_U64_GE V0003, V0002          GOTO L020_OK

    ; x = i
    STORE_U64 V0004, V0003

    ; j = n - 1 (signed)
    SUB_S64 V0005, ARG1, 1

L020_J_LOOP:
    ; if j < 0: finished digits
    IF_S64_LT V0005, 0              GOTO L020_WRITE

    ; digit = x & 63
    AND_U64 V0006, V0004, D0007

    ; ch = alphabet[digit]
    LOAD_U8_INDEX V0007, ARG2, V0006      ; V0007 = *(alphabet + digit)

    ; BUF[j] = ch
    STORE_U8_INDEX V0010, V0005, V0007      ; *(BUF + j) = ch

    ; x >>= 6
    SHR_U64 V0004, V0004, 6

    ; j--
    SUB_S64 V0005, V0005, 1
    GOTO L020_J_LOOP

L020_WRITE:
    ; write BUF[0..n-1]
    CALL 502 ARG3, &V0010, ARG1      -> V0007
    IF_U8_EQ V0007, 0                 GOTO L020_FAIL

    ; write '\n'
    CALL 503 ARG3, D0008            -> V0007
    IF_U8_EQ V0007, 0                 GOTO L020_FAIL
```

```

; i++
ADD_U64    V0003, V0003, 1
GOTO L020_I_LOOP

L020_FAIL:
RETURN_U8 0

L020_OK:
RETURN_U8 1
ENDSUB

; =====
; MAIN PROGRAM DECK (A-0 "sequence of subroutines + arguments")
; Mirrors: main() in the assembly
; =====

MAIN:
; file = fopen("system_safe.txt", "w")
CALL 500  D0002, D0003           -> V0001

; if file == 0: perror("Error opening file"); exit(1)
IF_PTR_EQ V0001, 0               GOTO M_OPEN_FAIL

; ok = GEN_PERMUTATIONS(N, ALPHABET, file)
CALL 020  D0005, D0001, V0001     -> V0007

; close file (best-effort)
CALL 501  V0001                 -> V0006

; if ok == 0: exit(1) else exit(0)
IF_U8_EQ  V0007, 0               GOTO M_EXIT_FAIL
CALL 999  0
HALT

M_OPEN_FAIL:
CALL 520  D0004
CALL 999  1
HALT

M_EXIT_FAIL:
CALL 999  1
HALT
ENDMAIN

```

## [2] Architecture/A\_plus.txt

- **Bytes:** 7804
- **Type:** text

```

;;;;;;;;;;;;;;;;;;
;; A+ MONOLITHIC SCRIPT
;; Name: system_safe_generator.aplus
;; Source: assembly.txt  (safe_uint64_power, generate_permutations, file sink,
main)
;;;;;;;;;;;;;;;;;;

A+::UNIT "system_safe_generator"
A+::VERSION 1.0

;;;;;;;;;;;;;;;;;;
;; Primitive Types
;;;;;;;;;;;;;;;;;;

TYPE u8    = UINT(8)
TYPE u32   = UINT(32)
TYPE u64   = UINT(64)
TYPE u128  = UINT(128)
TYPE i32   = SINT(32)
TYPE i64   = SINT(64)
TYPE bool  = ENUM { false=0, true=1 }

TYPE ptr[T] = POINTER(T)

;;;;;;;;;;;;;;;;;;
;; Minimal OS / C-ABI Surface (conceptual intrinsics)
;; (A+ runtimes typically provide equivalents; keep names stable for clarity.)
;;;;;;;;;;;;;;;;;;

TYPE FileHandle = ptr[opaque]

INTRINSIC OS.fopen(path: ptr[u8], mode: ptr[u8]) -> FileHandle
INTRINSIC OS.fwrite(buf: ptr[u8], size: u64, count: u64, fh: FileHandle) -> u64
INTRINSIC OS.fputc(ch: i32, fh: FileHandle) -> i32
INTRINSIC OS.fclose(fh: FileHandle) -> i32
INTRINSIC OS.perror(msg: ptr[u8]) -> void

INTRINSIC MEM.zero(buf: ptr[u8], n: u64) -> void

;;;;;;;;;;;;;;;;;;
;; Constants (matches assembly .ascii/.string payloads)
;;;;;;;;;;;;;;;;;;

CONST ALPHABET_STR : BYTES =
    "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 \n"
;; Length must be exactly 64:
;; 26 lower + 26 upper + 10 digits + space + '\n' = 64.

CONST MODE_WRITE : BYTES = "w"
CONST OUT_PATH   : BYTES = "system_safe.txt"
CONST ERR_OPEN   : BYTES = "Error opening file"

CONST U64_MAX : u128 = 18446744073709551615 ;; 2^64 - 1

```

```
;;;;;;
;; OutputSink (layout mirrors the idea in assembly: ctx + function pointers)
;;;;;;

STRUCT OutputSink {
    ctx      : FileHandle
    write    : fn(ctx: FileHandle, data: ptr[u8], n: u64) -> bool
    write_char : fn(ctx: FileHandle, ch: u8) -> bool
}

;;;;;;
;; Helpers: safe u64 multiply (models x86 MUL overflow test)
;;;;;;

FN safe_mul_u64(a: u64, b: u64, out: ref u64) -> bool
{
    LET prod : u128 = (u128)a * (u128)b
    IF prod > U64_MAX THEN
        RETURN false
    ENDIF
    out = (u64)prod
    RETURN true
}

;;;;;;
;; safe_uint64_power(base, exp, out*) -> bool
;; Assembly behavior:
;;   *out = 1
;;   repeat exp times: if overflow return 0 else *out *= base
;;   return 1 on success
;;;;;;

FN safe_uint64_power(base: u64, exp: u64, out: ref u64) -> bool
{
    out = 1
    LET i : u64 = 0

    WHILE i < exp DO
        LET tmp : u64 = out
        IF NOT safe_mul_u64(tmp, base, out) THEN
            RETURN false
        ENDIF
        i = i + 1
    ENDWHILE

    RETURN true
}

;;;;;;
;; file_sink_write(ctx, data, n) -> bool
;; Assembly uses fwrite(data, 1, n, FILE*) and checks written == n
;;;;;;
```

```

FN file_sink_write(ctx: FileHandle, data: ptr[u8], n: u64) -> bool
{
    LET written : u64 = OS.fwrite(data, 1, n, ctx)
    RETURN (written == n)
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; file_sink_write_char(ctx, ch) -> bool
;; Assembly uses fputc(ch, FILE*) and checks != -1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

FN file_sink_write_char(ctx: FileHandle, ch: u8) -> bool
{
    LET rc : i32 = OS.fputc((i32)ch, ctx)
    RETURN (rc != -1)
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; file_sink_init(sink*, path) -> bool
;; Assembly: fopen(path,"w"); on success sets sink.ctx and fn pointers
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

FN file_sink_init(sink: ref OutputSink, path: BYTES) -> bool
{
    LET fh : FileHandle = OS.fopen(&path[0], &MODE_WRITE[0])
    IF fh == NULL THEN
        RETURN false
    ENDIF

    sink.ctx      = fh
    sink.write    = file_sink_write
    sink.write_char = file_sink_write_char
    RETURN true
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; file_sink_close(sink*) -> void
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

FN file_sink_close(sink: ref OutputSink) -> void
{
    IF sink.ctx != NULL THEN
        OS.fclose(sink.ctx)
        sink.ctx = NULL
    ENDIF
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; generate_permutations(n, sink*) -> bool
;; Assembly behavior:
;;   total = 64^n (checked with safe_uint64_power)
;;   for i in [0, total):
;;       tmp = i
;;       idx = n-1 down to 0:

```

```
;;      digit = tmp & 63
;;      buf[idx] = ALPHABET[digit]
;;      tmp >>= 6
;;      sink.write(buf, n) must succeed
;;      sink.write_char(10) must succeed
;;      return 1 on full success else 0
;;;;;;;;;;;;;;;;
FN generate_permutations(n: u64, sink: ref OutputSink) -> bool
{
    ;; total permutations: 64^n
    LET total : u64 = 0
    IF NOT safe_uint64_power(64, n, total) THEN
        RETURN false
    ENDIF

    ;; buffer (monolithic, fixed upper bound; we only use first n bytes)
    ;; assembly uses a small stack buffer; here we cap at 64 for safety.
    IF n > 64 THEN
        RETURN false
    ENDIF

    LET buf : ARRAY[u8, 64]
    MEM.zero(&buf[0], 64)

    LET i : u64 = 0
    WHILE i < total DO
        LET tmp : u64 = i
        LET idx : i64 = (i64)n - 1

        WHILE idx >= 0 DO
            LET digit : u64 = tmp & 63
            buf[(u64)idx] = (u8)ALPHABET_STR[digit]
            tmp = tmp >> 6
            idx = idx - 1
        ENDWHILE

        ;; write n bytes
        IF NOT sink.write(sink.ctx, &buf[0], n) THEN
            RETURN false
        ENDIF

        ;; then write '\n' (LF = 10), matching assembly
        IF NOT sink.write_char(sink.ctx, (u8)10) THEN
            RETURN false
        ENDIF

        i = i + 1
    ENDWHILE

    RETURN true
}
;;;;;;;;;;;;;;;
```

```

;; main
;; Assembly:
;;   n = 4
;;   init sink("system_safe.txt") else perror("Error opening file") and exit 1
;;   ok = generate_permutations(4, &sink)
;;   close sink
;;   return 0 if ok else 1
;;;;;;;;;;;;;;;

FN main() -> i32
{
    LET n : u64 = 4

    LET sink : OutputSink
    sink.ctx      = NULL
    sink.write    = file_sink_write
    sink.write_char = file_sink_write_char

    IF NOT file_sink_init(sink, OUT_PATH) THEN
        OS.perror(&ERR_OPEN[0])
        RETURN 1
    ENDIF

    LET ok : bool = generate_permutations(n, sink)

    file_sink_close(sink)

    IF NOT ok THEN
        RETURN 1
    ENDIF

    RETURN 0
}

A++:ENTRY main

```

### [3] Architecture/A\_plus\_plus.txt

- **Bytes:** 6608
- **Type:** text

```

/*
=====
A++ Monolith: system_safe_permutations.a++
=====

Source-equivalent translation of assembly.txt. :contentReference[oaicite:1]
{index=1}

What it does:

```

- ALPHABET is 64 symbols: a-z A-Z 0-9 ' ' '\n'
- For n = 4 (main), enumerates all 64^n codes.
- Each code is written as raw bytes (NOT NUL-terminated), then a '\n'.
- NOTE: because '\n' is itself a symbol in ALPHABET, some generated codes contain newline bytes \*inside\* the code, which will visually break lines in the output file (this matches the assembly behavior).

Output size for n=4:

- $64^4 = 16,777,216$  records (huge).

\*/

```
/* =====
```

```
A++: Core scalar types
```

```
===== */
```

```
type U8    = unsigned_8;
type U32   = unsigned_32;
type U64   = unsigned_64;
type U128  = unsigned_128;
type I32   = signed_32;
type Bool  = unsigned_8; // 0 or 1
```

```
const Bool FALSE = 0;
const Bool TRUE  = 1;
```

```
const U64 U64_MAX = 18446744073709551615;
```

```
/* =====
```

```
A++: Minimal FFI surface
```

```
=====
```

```
These are abstract bindings to the host C runtime.
```

\*/

```
opaque type FILE;
```

```
/* C stdio bindings */
```

```
extern func fopen(path: ptr<const U8>, mode: ptr<const U8>) -> ptr<FILE>;
extern func fclose(f: ptr<FILE>) -> I32;
extern func fwrite(buf: ptr<const U8>, size: U64, count: U64, f: ptr<FILE>) -> U64;
extern func fputc(ch: I32, f: ptr<FILE>) -> I32;
extern func perror(msg: ptr<const U8>) -> void;
```

```
/* =====
```

```
A++: Memory helpers
```

```
===== */
```

```
extern func malloc(n: U64) -> ptr<U8>;
extern func free(p: ptr<U8>) -> void;
```

```
/* =====
```

```
A++: OutputSink interface
```

```
===== */
```

```

type WriteFn      = func(ctx: ptr<void>, buf: ptr<const U8>, len: U64) -> Bool;
type WriteCharFn = func(ctx: ptr<void>, ch: U8) -> Bool;

struct OutputSink {
    ctx: ptr<void>;
    write: WriteFn;
    write_char: WriteCharFn;
}

/* =====
   Embedded Alphabet (64 bytes)
   Matches:
   "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567"
   "89 \n"
===== */

const U8 ALPHABET[64] = [
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
    'v', 'w', 'x', 'y', 'z',
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
    'V', 'W', 'X', 'Y', 'Z',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '\n'
];

/* =====
   Safe arithmetic
===== */

func u64_mul_overflow(a: U64, b: U64, out: ptr<U64>) -> Bool {
    // Implements the same intent as the assembly's MUL + overflow flag test.
    // Here, we use a widened multiplication and compare against U64_MAX.
    let wide: U128 = (U128)a * (U128)b;
    if wide > (U128)U64_MAX {
        return TRUE; // overflow occurred
    }
    *out = (U64)wide;
    return FALSE; // no overflow
}

/*
safe_uint64_power(base, exp, out):
    out := base^exp, returning TRUE on success, FALSE on overflow.
    This matches the assembly routine that multiplies exp times, checking overflow.
*/
func safe_uint64_power(base: U64, exp: U64, out: ptr<U64>) -> Bool {
    *out = 1;
    let i: U64 = 0;

    while i < exp {
        let tmp: U64 = 0;
        let ov: Bool = u64_mul_overflow(*out, base, &tmp);
        if ov == TRUE {
            return FALSE;
        }
        *out *= base;
        i += 1;
    }
}

```

```
        return FALSE;
    }
    *out = tmp;
    i = i + 1;
}

return TRUE;
}

/* =====
File sink implementation
===== */

func file_sink_write(ctx: ptr<void>, buf: ptr<const U8>, len: U64) -> Bool {
let f: ptr<FILE> = (ptr<FILE>)ctx;

// fwrite(buf, 1, len, f) must return len to succeed.
let wrote: U64 = fwrite(buf, 1, len, f);
if wrote == len { return TRUE; }
return FALSE;
}

func file_sink_write_char(ctx: ptr<void>, ch: U8) -> Bool {
let f: ptr<FILE> = (ptr<FILE>)ctx;

// fputc returns -1 on error.
let rc: I32 = fputc((I32)ch, f);
if rc != -1 { return TRUE; }
return FALSE;
}

func file_sink_init(sink: ptr<OutputSink>, path: ptr<const U8>) -> Bool {
const U8 MODE_W[2] = ['w', 0];

let f: ptr<FILE> = fopen(path, &MODE_W[0]);
if f == (ptr<FILE>)0 {
    return FALSE;
}

sink->ctx = (ptr<void>)f;
sink->write = file_sink_write;
sink->write_char = file_sink_write_char;
return TRUE;
}

func file_sink_close(sink: ptr<OutputSink>) -> void {
if sink == (ptr<OutputSink>)0 { return; }
if sink->ctx == (ptr<void>)0 { return; }

let f: ptr<FILE> = (ptr<FILE>)sink->ctx;
fclose(f);
sink->ctx = (ptr<void>)0;
}
```

```
/* =====
Permutation generator
=====
generate_permutations(n, sink):
    total = 64^n
    for i in [0..total-1]:
        encode i in base64 digits (6-bit chunks) into n bytes:
        idx = (tmp & 63)
        tmp >>= 6
        write n bytes then '\n'
*/
func generate_permutations(n: U64, sink: ptr<OutputSink>) -> Bool {
    let total: U64 = 0;

    if safe_uint64_power(64, n, &total) == FALSE {
        return FALSE;
    }

    // Allocate exactly n bytes (no NUL terminator), matching fwrite usage.
    let buf: ptr<U8> = malloc(n);
    if buf == (ptr<U8>)0 {
        return FALSE;
    }

    let i: U64 = 0;
    while i < total {

        let tmp: U64 = i;

        // Fill from end to start: pos = n-1 down to 0
        // Matches the assembly decrementing an index and writing into [rbp-50 + pos].
        let pos: I32 = (I32)(n - 1);

        while pos >= 0 {
            let idx: U64 = (tmp & 63);
            buf[(U64)pos] = ALPHABET[idx];
            tmp = (tmp >> 6);
            pos = pos - 1;
        }

        if sink->write(sink->ctx, buf, n) == FALSE {
            free(buf);
            return FALSE;
        }

        if sink->write_char(sink->ctx, (U8)10) == FALSE { // '\n'
            free(buf);
            return FALSE;
        }

        i = i + 1;
    }
}
```

```

    free(buf);
    return TRUE;
}

/* =====
Program entrypoint
===== */

func main() -> I32 {
    const U8 OUT_PATH[] = "system_safe.txt\0";
    const U8 ERR_MSG[] = "Error opening file\0";

    let sink: OutputSink;

    // In the assembly, n is set to 4.
    let n: U64 = 4;

    if file_sink_init(&sink, &OUT_PATH[0]) == FALSE {
        perror(&ERR_MSG[0]);
        return 1;
    }

    let ok: Bool = generate_permutations(n, &sink);

    file_sink_close(&sink);

    if ok == TRUE { return 0; }
    return 1;
}

```

#### [4] Architecture/A\_sharp\_axiom.txt

- **Bytes:** 6280
- **Type:** text

```

// A# (Axiom) Monolithic Script (C#-compatible .NET single-file)
// -----
// Behavior matches the attached program:
// - ALPHABET: 64 chars: a-z A-Z 0-9 space newline
// - safe_uint64_power(64, n, &outTotal) with overflow detection
// - generate_permutations(n, sink): for i in [0, 64^n]:
//     build n bytes by repeated: digit = tmp & 63; tmp >>= 6
//     store into buffer from right-to-left (pos = n-1..0)
//     sink.write(buffer, n); sink.write_char('\n')
// - main: n=4, output file "system_safe.txt"
//
// Notes:
// - This intentionally writes LF '\n' (byte 10) as in the assembly, not CRLF.
// - Because ALPHABET includes '\n', some generated "strings" contain newlines
//   inside them

```

```
// exactly like the original program, producing additional line breaks in the
output.

using System;
using System.IO;
using System.Text;

public static class AxiomProgram
{
    // 64-character alphabet (matches the assembly exactly):
    // "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 \n"
    private static readonly byte[] ALPHABET = Encoding.ASCII.GetBytes(
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 \n"
    );

    // OutputSink equivalent: ctx + function pointers (modeled as virtual methods
    here).
    private abstract class OutputSink : IDisposable
    {
        public abstract bool Write(ReadOnlySpan<byte> data);
        public abstract bool WriteChar(byte c);
        public abstract void Dispose();
    }

    // File-backed sink (fopen/fwrite/fputc/fclose analog).
    private sealed class FileSink : OutputSink
    {
        private FileStream? _fs;

        public static bool Init(out FileSink sink, string path)
        {
            sink = new FileSink();
            try
            {
                // "w" mode equivalent (truncate/create).
                sink._fs = new FileStream(
                    path,
                    FileMode.Create,
                    FileAccess.Write,
                    FileShare.Read,
                    bufferSize: 1 << 20, // 1 MiB buffer for throughput
                    useAsync: false
                );
                return true;
            }
            catch
            {
                sink.Dispose();
                return false;
            }
        }

        public override bool Write(ReadOnlySpan<byte> data)
        {

```

```
try
{
    if (_fs is null) return false;
    _fs.Write(data);
    return true;
}
catch
{
    return false;
}
}

public override bool WriteChar(byte c)
{
    try
    {
        if (_fs is null) return false;
        _fs.WriteByte(c);
        return true;
    }
    catch
    {
        return false;
    }
}

public override void Dispose()
{
    try { _fs?.Dispose(); } catch { /* swallow */ }
    _fs = null;
}

// safe_uint64_power(unsigned long base, unsigned long exp, unsigned long*
out)
// Returns: true on success, false on overflow (matches assembly: eax=1
success, eax=0 fail).
private static bool SafeUInt64Power(ulong @base, ulong exp, out ulong result)
{
    result = 1UL;

    // exp loop like the assembly (i from 0 while i<exp).
    for (ulong i = 0; i < exp; i++)
    {
        // Detect overflow for result *= base
        // (unsigned multiply overflow check).
        if (@base != 0 && result > (ulong.MaxValue / @base))
            return false;

        result *= @base;
    }

    return true;
}
```

```
// generate_permutations(unsigned long n, OutputSink* sink)
// Returns: true on success, false if sink write fails or overflow in power.
private static bool GeneratePermutations(ulong n, OutputSink sink)
{
    // total = 64^n (safe)
    if (!SafeUInt64Power(64UL, n, out ulong total))
        return false;

    // Buffer corresponds to the stack buffer used in the assembly.
    // We write exactly n bytes to sink, then a '\n' char via WriteChar(10).
    // Use a fixed byte[] for speed; fill per iteration.
    if (n > int.MaxValue) return false; // sanity (won't happen here)
    int len = (int)n;
    byte[] buf = new byte[len];

    for (ulong i = 0; i < total; i++)
    {
        ulong tmp = i;
        long pos = len - 1; // assembly: starts at n-1 and decrements to 0

        while (pos >= 0)
        {
            // digit = tmp & 63
            int digit = (int)(tmp & 63UL);
            buf[pos] = ALPHABET[digit];

            // tmp >>= 6
            tmp >>= 6;

            pos--;
        }

        // sink.write(ctx, buf, n)
        if (!sink.Write(buf))
            return false;

        // sink.write_char(ctx, 10)
        if (!sink.WriteChar((byte)10))
            return false;
    }

    return true;
}

// main (matches the assembly's behavior):
// - output path: "system_safe.txt"
// - n = 4
public static int Main(string[] args)
{
    const string defaultPath = "system_safe.txt";
    const ulong defaultN = 4;

    // Allow optional overrides (still monolithic; defaults match original):
```

```

// args[0] = output path, args[1] = n
string path = (args.Length >= 1 && !string.IsNullOrWhiteSpace(args[0])) ?
args[0] : defaultPath;
ulong n = defaultN;

if (args.Length >= 2 && ulong.TryParse(args[1], out ulong parsed))
    n = parsed;

if (!FileSink.Init(out FileSink sink, path))
{
    Console.Error.WriteLine("Error opening file");
    return 1;
}

bool ok;
try
{
    ok = GeneratePermutations(n, sink);
}
finally
{
    sink.Dispose();
}

return ok ? 0 : 1;
}
}

```

## [5] Architecture/A\_sharp\_dot-NET.txt

- **Bytes:** 6119
- **Type:** text

```

-- system_safe.adb
-- Monolithic A# (.NET) / Ada program equivalent to the provided assembly.
-- Writes all base-64 "digits" (using the given 64-char alphabet) for length N
-- into "system_safe.txt", one permutation per line.

with Ada.Characters.Latin_1;
with Ada.Command_Line;
with Ada.Text_IO;
with Ada.Streams;
with Ada.Streams.Stream_IO;
with Interfaces;

procedure System_Safe is
    use Interfaces;
    use Ada.Characters.Latin_1;

    -- ALPHABET:

```

```
-- "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 \n"
Alphabet : constant String :=
    "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 " & LF;

Default_Out_Path : constant String := "system_safe.txt";
Err_Open_Msg      : constant String := "Error opening file";

type Output_Sink is record
    File      : Ada.Streams.Stream_IO.File_Type;
    Is_Open   : Boolean := False;
end record;

function File_Sink_Init (Sink : in out Output_Sink; Path : String) return Boolean is
begin
    Ada.Streams.Stream_IO.Create
        (File => Sink.File,
         Mode => Ada.Streams.Stream_IO.Out_File,
         Name => Path);
    Sink.Is_Open := True;
    return True;
exception
    when others =>
        Sink.Is_Open := False;
        return False;
end File_Sink_Init;

procedure File_Sink_Close (Sink : in out Output_Sink) is
begin
    if Sink.Is_Open then
        Ada.Streams.Stream_IO.Close (Sink.File);
        Sink.Is_Open := False;
    end if;
exception
    when others =>
        null;
end File_Sink_Close;

function File_Sink_Write (Sink : in out Output_Sink; Data : String) return Boolean is
    use Ada.Streams;
    use Ada.Streams.Stream_IO;
    Buf : Stream_Element_Array (1 .. Stream_Element_Offset (Data'Length));
    J   : Stream_Element_Offset := 1;
begin
    if not Sink.Is_Open then
        return False;
    end if;

    for I in Data'Range loop
        Buf (J) := Stream_Element (Character'Pos (Data (I)));
        J := J + 1;
    end loop;
```

```
    Write (Sink.File, Buf);
    return True;
exception
  when others =>
    return False;
end File_Sink_Write;

function File_Sink_Write_Char (Sink : in out Output_Sink; C : Character) return Boolean is
  use Ada.Streams;
  use Ada.Streams.Stream_IO;
  Buf : Stream_Element_Array (1 .. 1);
begin
  if not Sink.Is_Open then
    return False;
  end if;

  Buf (1) := Stream_Element (Character'Pos (C));
  Write (Sink.File, Buf);
  return True;
exception
  when others =>
    return False;
end File_Sink_Write_Char;

-- safe_uint64_power(base, exp, *out) -> Boolean success (no overflow)
function Safe_Uint64_Power
  (Base      : Unsigned_64;
   Exp       : Unsigned_64;
   Result    : out Unsigned_64) return Boolean
is
  R : Unsigned_64 := 1;
  I : Unsigned_64 := 0;
begin
  Result := 1;

  while I < Exp loop
    if Base = 0 then
      R := 0;
    else
      -- overflow check: R * Base must fit in Unsigned_64
      if R > Unsigned_64'Last / Base then
        Result := 0;
        return False;
      end if;
      R := R * Base;
    end if;

    I := I + 1;
  end loop;

  Result := R;
  return True;
end Safe_Uint64_Power;
```

```

-- generate_permutations(len, sink) -> Boolean
function Generate_Permutations
  (Len : Natural;
   Sink : in out Output_Sink) return Boolean
is
  Total : Unsigned_64 := 0;
  I     : Unsigned_64 := 0;
begin
  if not Safe_Uint64_Power (Base => 64, Exp => Unsigned_64 (Len), Result =>
Total) then
    return False;
  end if;

  if Len = 0 then
    -- Matches the assembly behavior: total = 1; write empty string + '\n'
    return File_Sink_Write_Char (Sink, LF);
  end if;

declare
  Buffer : String (1 .. Len);
begin
  while I < Total loop
    declare
      Tmp : Unsigned_64 := I;
    begin
      -- Fill from right to left: base-64 digits (Tmp & 63), then Tmp >=
6
      for Pos in reverse Buffer'Range loop
        declare
          Idx : Natural := Natural (Tmp and 63); -- 0..63
        begin
          Buffer (Pos) := Alphabet (Alphabet'First + Idx);
        end;
        Tmp := Shift_Right (Tmp, 6);
      end loop;
    end;
    if not File_Sink_Write (Sink, Buffer) then
      return False;
    end if;

    if not File_Sink_Write_Char (Sink, LF) then
      return False;
    end if;

    I := I + 1;
  end loop;
end;

return True;
end Generate_Permutations;

-- main equivalents

```

```

Sink      : Output_Sink;
Len       : Natural := 4;
Out_Path : String  := Default_Out_Path;
Ok        : Boolean := False;

begin
  -- Optional CLI override:
  --  arg1 = length (Natural), arg2 = output path
  if Ada.Command_Line.Argument_Count >= 1 then
    declare
      use Ada.Text_IO;
    begin
      Len := Natural'Value (Ada.Command_Line.Argument (1));
    exception
      when others =>
        -- If parse fails, keep default
        null;
    end;
  end if;

  if Ada.Command_Line.Argument_Count >= 2 then
    Out_Path := Ada.Command_Line.Argument (2);
  end if;

  if not File_Sink_Init (Sink, Out_Path) then
    Ada.Text_IO.Put_Line (Ada.Text_IO.Standard_Error, Err_Open_Msg);
    Ada.Command_Line.Set_Exit_Status (Ada.Command_Line.Failure);
    return;
  end if;

  Ok := Generate_Permutations (Len, Sink);

  File_Sink_Close (Sink);

  if not Ok then
    Ada.Command_Line.Set_Exit_Status (Ada.Command_Line.Failure);
  else
    Ada.Command_Line.Set_Exit_Status (Ada.Command_Line.Success);
  end if;
end System_Safe;

```

## [6] Architecture/assembly.txt

- **Bytes:** 7299
- **Type:** text

ALPHABET:

```

.ascii  "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567"
.ascii  "89 \n"

```

```
safe_uint64_power(unsigned long, unsigned long, unsigned long*):
```

```
push    rbp
mov     rbp, rsp
mov     QWORD PTR [rbp-24], rdi
mov     QWORD PTR [rbp-32], rsi
mov     QWORD PTR [rbp-40], rdx
mov     rax, QWORD PTR [rbp-40]
mov     QWORD PTR [rax], 1
mov     QWORD PTR [rbp-8], 0
jmp     .L2

.L7:
mov     rax, QWORD PTR [rbp-40]
mov     rdx, QWORD PTR [rax]
mov     ecx, 0
mov     rax, rdx
mul    QWORD PTR [rbp-24]
jno    .L3
mov     ecx, 1

.L3:
mov     rax, rcx
test   rax, rax
je     .L5
mov     eax, 0
jmp     .L6

.L5:
mov     rax, QWORD PTR [rbp-40]
mov     rax, QWORD PTR [rax]
imul   rax, QWORD PTR [rbp-24]
mov     rdx, rax
mov     rax, QWORD PTR [rbp-40]
mov     QWORD PTR [rax], rdx
add    QWORD PTR [rbp-8], 1

.L2:
mov     rax, QWORD PTR [rbp-8]
cmp    rax, QWORD PTR [rbp-32]
jb    .L7
mov     eax, 1

.L6:
pop    rbp
ret

generate_permutations(unsigned long, OutputSink*):
push    rbp
mov     rbp, rsp
sub    rsp, 80
mov     QWORD PTR [rbp-72], rdi
mov     QWORD PTR [rbp-80], rsi
lea    rdx, [rbp-40]
mov     rax, QWORD PTR [rbp-72]
mov     rsi, rax
mov     edi, 64
call    safe_uint64_power(unsigned long, unsigned long, unsigned long*)
xor    eax, 1
test   al, al
je     .L9
mov     eax, 0
```

```
        jmp    .L17
.L9:
    mov    QWORD PTR [rbp-8], 0
    jmp    .L11
.L16:
    mov    rax, QWORD PTR [rbp-8]
    mov    QWORD PTR [rbp-16], rax
    mov    rax, QWORD PTR [rbp-72]
    sub    rax, 1
    mov    QWORD PTR [rbp-24], rax
    jmp    .L12
.L13:
    mov    rax, QWORD PTR [rbp-16]
    and    eax, 63
    mov    QWORD PTR [rbp-32], rax
    mov    rax, QWORD PTR [rbp-32]
    add    rax, OFFSET FLAT:ALPHABET
    movzx  eax, BYTE PTR [rax]
    lea    rcx, [rbp-50]
    mov    rdx, QWORD PTR [rbp-24]
    add    rdx, rcx
    mov    BYTE PTR [rdx], al
    shr    QWORD PTR [rbp-16], 6
    sub    QWORD PTR [rbp-24], 1
.L12:
    cmp    QWORD PTR [rbp-24], 0
    jns    .L13
    mov    rax, QWORD PTR [rbp-80]
    mov    r8, QWORD PTR [rax+8]
    mov    rax, QWORD PTR [rbp-80]
    mov    rax, QWORD PTR [rax]
    mov    rdx, QWORD PTR [rbp-72]
    lea    rcx, [rbp-50]
    mov    rsi, rcx
    mov    rdi, rax
    call   r8
    xor    eax, 1
    test   al, al
    je     .L14
    mov    eax, 0
    jmp    .L17
.L14:
    mov    rax, QWORD PTR [rbp-80]
    mov    rdx, QWORD PTR [rax+16]
    mov    rax, QWORD PTR [rbp-80]
    mov    rax, QWORD PTR [rax]
    mov    esi, 10
    mov    rdi, rax
    call   rdx
    xor    eax, 1
    test   al, al
    je     .L15
    mov    eax, 0
    jmp    .L17
```

```
.L15:
    add    QWORD PTR [rbp-8], 1
.L11:
    mov    rax, QWORD PTR [rbp-40]
    cmp    QWORD PTR [rbp-8], rax
    jb     .L16
    mov    eax, 1
.L17:
    leave
    ret
file_sink_write(void*, char const*, unsigned long):
    push   rbp
    mov    rbp, rsp
    sub    rsp, 48
    mov    QWORD PTR [rbp-24], rdi
    mov    QWORD PTR [rbp-32], rsi
    mov    QWORD PTR [rbp-40], rdx
    mov    rax, QWORD PTR [rbp-24]
    mov    QWORD PTR [rbp-8], rax
    mov    rcx, QWORD PTR [rbp-8]
    mov    rdx, QWORD PTR [rbp-40]
    mov    rax, QWORD PTR [rbp-32]
    mov    esi, 1
    mov    rdi, rax
    call   fwrite
    cmp    QWORD PTR [rbp-40], rax
    sete   al
    leave
    ret
file_sink_write_char(void*, char):
    push   rbp
    mov    rbp, rsp
    sub    rsp, 32
    mov    QWORD PTR [rbp-24], rdi
    mov    eax, esi
    mov    BYTE PTR [rbp-28], al
    mov    rax, QWORD PTR [rbp-24]
    mov    QWORD PTR [rbp-8], rax
    movsx  eax, BYTE PTR [rbp-28]
    mov    rdx, QWORD PTR [rbp-8]
    mov    rsi, rdx
    mov    edi, eax
    call   fputc
    cmp    eax, -1
    setne  al
    leave
    ret
.LC0:
    .string "w"
file_sink_init(OutputSink*, char const*):
    push   rbp
    mov    rbp, rsp
    sub    rsp, 32
    mov    QWORD PTR [rbp-24], rdi
```

```
    mov    QWORD PTR [rbp-32], rsi
    mov    rax, QWORD PTR [rbp-32]
    mov    esi, OFFSET FLAT:.LC0
    mov    rdi, rax
    call   fopen
    mov    QWORD PTR [rbp-8], rax
    cmp    QWORD PTR [rbp-8], 0
    jne   .L23
    mov    eax, 0
    jmp   .L24

.L23:
    mov    rax, QWORD PTR [rbp-24]
    mov    rdx, QWORD PTR [rbp-8]
    mov    QWORD PTR [rax], rdx
    mov    rax, QWORD PTR [rbp-24]
    mov    QWORD PTR [rax+8], OFFSET FLAT:file_sink_write(void*, char const*,  
unsigned long)
    mov    rax, QWORD PTR [rbp-24]
    mov    QWORD PTR [rax+16], OFFSET FLAT:file_sink_write_char(void*, char)
    mov    eax, 1

.L24:
    leave
    ret

file_sink_close(OutputSink*):
    push   rbp
    mov    rbp, rsp
    sub    rsp, 16
    mov    QWORD PTR [rbp-8], rdi
    cmp    QWORD PTR [rbp-8], 0
    je     .L27
    mov    rax, QWORD PTR [rbp-8]
    mov    rax, QWORD PTR [rax]
    test   rax, rax
    je     .L27
    mov    rax, QWORD PTR [rbp-8]
    mov    rax, QWORD PTR [rax]
    mov    rdi, rax
    call   fclose
    mov    rax, QWORD PTR [rbp-8]
    mov    QWORD PTR [rax], 0

.L27:
    nop
    leave
    ret

.LC1:
    .string "system_safe.txt"

.LC2:
    .string "Error opening file"

main:
    push   rbp
    mov    rbp, rsp
    sub    rsp, 48
    mov    QWORD PTR [rbp-8], 4
    lea    rax, [rbp-48]
```

```

        mov    esi, OFFSET FLAT:.LC1
        mov    rdi, rax
        call   file_sink_init(OutputSink*, char const*)
        xor    eax, 1
        test   al, al
        je     .L29
        mov    edi, OFFSET FLAT:.LC2
        call   perror
        mov    eax, 1
        jmp   .L32

.L29:
        lea    rax, [rbp-48]
        mov    rsi, rax
        mov    edi, 4
        call   generate_permutations(unsigned long, OutputSink*)
        mov    BYTE PTR [rbp-9], al
        lea    rax, [rbp-48]
        mov    rdi, rax
        call   file_sink_close(OutputSink*)
        movzx eax, BYTE PTR [rbp-9]
        xor    eax, 1
        test   al, al
        je     .L31
        mov    eax, 1
        jmp   .L32

.L31:
        mov    eax, 0

.L32:
        leave
        ret

```

## [7] Architecture/C.txt

- **Bytes:** 4671
- **Type:** text

```

/**
 * @file main.c
 * @brief Permutation Generator in C, architected to safety-critical standards.
 *
 * @author Dominic Alexander Cooper (Original Algorithm)
 * @author Gemini (Architectural Refactoring)
 *
 * @details
 * This program generates all possible character combinations for a given length,
 * based on a predefined character set. It is a complete rewrite of the original
 * concept to align with principles of safety-critical software design.
 */
#include <stdio.h>
#include <stdint.h> // For fixed-width integer types like uint64_t
#include <stdbool.h> // For bool type

```

```
#include <limits.h> // For UINT64_MAX
//=====
// 1. CONFIGURATION AND DATA DEFINITIONS
//=====

#define MAX_PERMUTATION_LENGTH 10
static const char ALPHABET[] = {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
    'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
    'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '\n'
};

static const uint64_t ALPHABET_SIZE = sizeof(ALPHABET) / sizeof(ALPHABET[0]);
//=====

// 2. ABSTRACT INTERFACE FOR OUTPUT (OutputSink)
//=====

typedef struct {
    void* context;
    bool (*write)(void* context, const char* buffer, uint64_t size);
    bool (*write_char)(void* context, char c);
} OutputSink;
//=====

// 3. CORE LOGIC (Generator)
//=====

static bool safe_uint64_power(uint64_t base, uint64_t exp, uint64_t* result) {
    *result = 1;
    for (uint64_t i = 0; i < exp; ++i) {
        if (*result > UINT64_MAX / base) {
            return false;
        }
        *result *= base;
    }
    return true;
}
static bool generate_permutations(uint64_t length, OutputSink* sink) {
    uint64_t num_combinations;
    if (!safe_uint64_power(ALPHABET_SIZE, length, &num_combinations)) {
        return false;
    }
    char current_perm[MAX_PERMUTATION_LENGTH];
    for (uint64_t i = 0; i < num_combinations; ++i) {
        uint64_t temp_row = i;
        for (int64_t j = length - 1; j >= 0; --j) {
            uint64_t char_index = temp_row % ALPHABET_SIZE;
            current_perm[j] = ALPHABET[char_index];
            temp_row /= ALPHABET_SIZE;
        }
        if (!sink->write(sink->context, current_perm, length)) {
            return false;
        }
        if (!sink->write_char(sink->context, '\n')) {
            return false;
        }
    }
}
```

```
    return true;
}

//=====
// 4. CONCRETE IMPLEMENTATION OF OUTPUTSINK (FileSink)
//=====

static bool file_sink_write(void* context, const char* buffer, uint64_t size) {
    FILE* p = (FILE*)context;
    return fwrite(buffer, sizeof(char), size, p) == size;
}

static bool file_sink_write_char(void* context, char c) {
    FILE* p = (FILE*)context;
    return fputc(c, p) != EOF;
}

static bool file_sink_init(OutputSink* sink, const char* filename) {
    FILE* p = fopen(filename, "w");
    if (p == NULL) {
        return false;
    }
    *sink = (OutputSink){
        .context = p,
        .write = file_sink_write,
        .write_char = file_sink_write_char
    };
    return true;
}

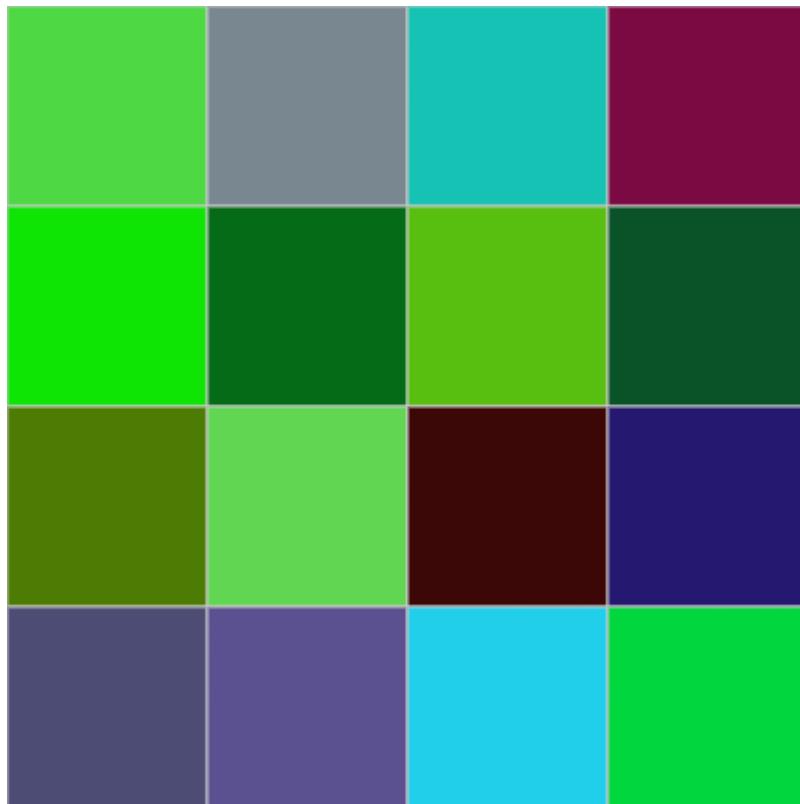
static void file_sink_close(OutputSink* sink) {
    if (sink && sink->context) {
        fclose((FILE*)sink->context);
        sink->context = NULL;
    }
}

//=====
// 5. SYSTEM ASSEMBLER (main)
//=====

int main(void) {
    const uint64_t permutation_length = 4;
    if (permutation_length == 0 || permutation_length > MAX_PERMUTATION_LENGTH) {
        return 1;
    }
    OutputSink file_sink;
    if (!file_sink_init(&file_sink, "system_safe.txt")) {
        perror("Error opening file");
        return 1;
    }
    bool success = generate_permutations(permutation_length, &file_sink);
    file_sink_close(&file_sink);
    if (!success) {
        return 1;
    }
    return 0;
}
```

## [8] Architecture/vector.png

- **Bytes:** 6320
- **Type:** png
- **Dimensions:** 400x400
- **Path (from doc):** ../Architecture/vector.png



## [9] Architecture/vector.txt

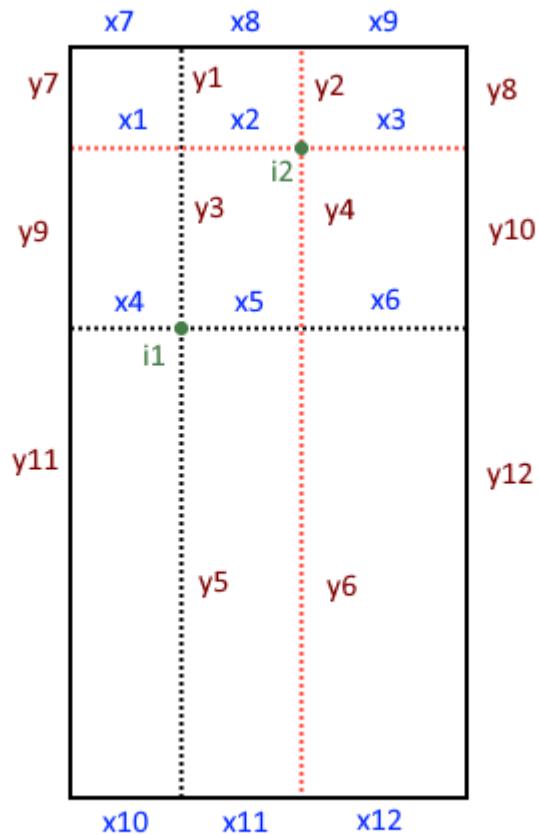
- **Bytes:** 139
- **Type:** text

```
5167173796481714261008063555097613404206315816081067639350124856411859386868023655  
54500030960501932215916054846
```

Principle that holds all.

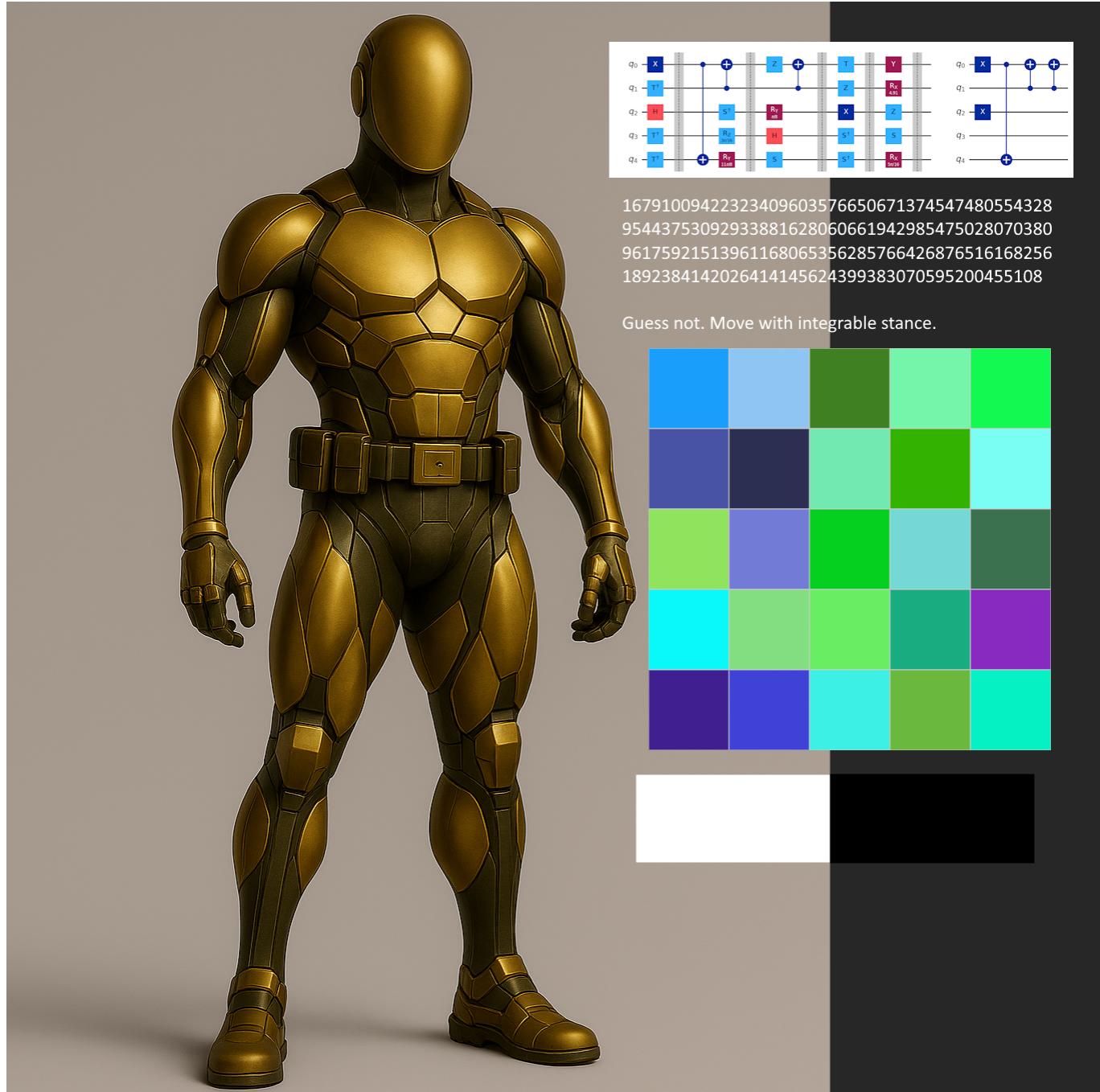
## [10] Art/Stone\_12.png

- **Bytes:** 12680
- **Type:** png
- **Dimensions:** 501x501
- **Path (from doc):** ../Art/Stone\_12.png



[11] Art/Stone\_39.png

- **Bytes:** 1903315
- **Type:** png
- **Dimensions:** 1361×1361
- **Path (from doc):** ../Art/Stone\_39.png



## [12] bundle\_documentation.py

- **Bytes:** 9214
- **Type:** text

```
#!/usr/bin/env python3
from __future__ import annotations

import argparse
import datetime as _dt
import os
import re
from pathlib import Path
from typing import Dict, Iterable, List, Optional, Tuple, Union
```

```
# ---- CONFIG ----
TEXT_EXTENSIONS = {
    ".css", ".cmd", ".js", ".php", ".hpp", ".cpp", ".md", ".py", ".txt", ".ps1",
    ".json", ".html"
}
IMAGE_EXTENSIONS = {".png"}

SPECIAL_FILERAMES = {".env", ".gitignore"} # extensionless-but-important

IGNORE_DIRS = {
    ".git", ".svn", ".hg",
    "node_modules", "vendor",
    "venv", ".venv", "__pycache__",
    "dist", "build",
    ".idea", ".vscode",
    "doc", # prevent bundling the bundle output folder itself
}

MAX_TEXT_FILE_BYTES = 2_000_000 # 2MB cap for *text* files

# ---- HELPERS ----
def is_probably_binary(path: Path, sample_size: int = 4096) -> bool:
    try:
        with path.open("rb") as f:
            chunk = f.read(sample_size)
            return b"\x00" in chunk
    except Exception:
        return True

def is_text_included(path: Path) -> bool:
    if path.name in SPECIAL_FILERAMES:
        return True
    return path.suffix.lower() in TEXT_EXTENSIONS

def is_png(path: Path) -> bool:
    return path.suffix.lower() == ".png"

def is_included_any(path: Path) -> bool:
    return is_text_included(path) or (path.suffix.lower() in IMAGE_EXTENSIONS)

def iter_included_paths(root: Path) -> Iterable[Path]:
    for dirpath, dirnames, filenames in os.walk(root):
        # prune ignored dirs
        dirnames[:] = [d for d in dirnames if d not in IGNORE_DIRS]

        for name in filenames:
            p = Path(dirpath) / name
            if p.is_file() and is_included_any(p):
                yield p
```

```
def read_text(path: Path) -> Tuple[str, str]:
    """
    Returns (content, note). note is "" when OK.
    """
    try:
        size = path.stat().st_size
        if size > MAX_TEXT_FILE_BYTES:
            return "", f"Skipped (too large): {size} bytes > {MAX_TEXT_FILE_BYTES}"
        if isProbablyBinary(path):
            return "", "Skipped (binary detected)"
        return path.read_text(encoding="utf-8", errors="replace"), ""
    except Exception as e:
        return "", f"Error reading: {e}"


def detect_code_lang(path: Path) -> str:
    ext = path.suffix.lower()
    return {
        ".py": "python",
        ".js": "javascript",
        ".ts": "typescript",
        ".json": "json",
        ".html": "html",
        ".css": "css",
        ".ps1": "powershell",
        ".cmd": "bat",
        ".cpp": "cpp",
        ".hpp": "cpp",
        ".php": "php",
        ".md": "markdown",
        ".txt": "text",
        "": "text",
    }.get(ext, "text")


def fenced_block(content: str, lang: str) -> str:
    """
    Create a fenced code block that won't break if the content contains ``
    already.
    """
    # Find the longest run of backticks in the content
    runs = [len(m.group(0)) for m in re.finditer(r"``+", content)]
    fence_len = max(runs) + 1 if runs else 3
    fence = `` * max(3, fence_len)
    return f"{fence}{lang}\n{content}\n{fence}\n"


def relpath_posix(from_dir: Path, to_path: Path) -> str:
    rel = os.path.relpath(to_path, start=from_dir)
    return Path(rel).as_posix()
```

```
def png_dimensions(path: Path) -> Optional[Tuple[int, int]]:
    """
    Parse PNG IHDR to get (width, height) without external dependencies.
    Returns None if unreadable/not PNG.
    """
    try:
        with path.open("rb") as f:
            header = f.read(24)
        if len(header) < 24:
            return None
        sig = header[:8]
        if sig != b"\x89PNG\r\n\x1a\n":
            return None
        chunk_type = header[12:16]
        if chunk_type != b"IHDR":
            return None
        w = int.from_bytes(header[16:20], "big")
        h = int.from_bytes(header[20:24], "big")
        return (w, h)
    except Exception:
        return None

# ----- TREE BUILD + RENDER -----
TreeNode = Dict[str, Union["TreeNode", None]] # None means leaf file

def build_tree(relpaths: List[Path]) -> TreeNode:
    root: TreeNode = {}
    for rp in relpaths:
        cur = root
        parts = rp.parts
        for i, part in enumerate(parts):
            is_last = (i == len(parts) - 1)
            if is_last:
                cur.setdefault(part, None)
            else:
                nxt = cur.get(part)
                if nxt is None:
                    cur[part] = {}
                cur = cur[part] # type: ignore[assignment]
    return root

def render_tree(tree: TreeNode, prefix: str = "") -> List[str]:
    """
    ASCII tree with deterministic ordering:
    directories first, then files, each group alphabetically.
    """
    lines: List[str] = []
    items = list(tree.items())

    def is_dir(item):
```

```
    _, v = item
    return isinstance(v, dict)

    dirs = sorted([it for it in items if is_dir(it)], key=lambda x: x[0].lower())
    files = sorted([it for it in items if not is_dir(it)], key=lambda x:
x[0].lower())
    ordered = dirs + files

    for idx, (name, node) in enumerate(ordered):
        last = (idx == len(ordered) - 1)
        branch = "└ " if last else "├ "
        lines.append(prefix + branch + name)

        if isinstance(node, dict):
            extension = "    " if last else "|   "
            lines.extend(render_tree(node, prefix + extension))

    return lines

# ----- MAIN OUTPUT -----
def create_doc_md(root: Path, outpath: Path) -> Path:
    files = sorted(iter_included_paths(root), key=lambda p:
p.relative_to(root).as_posix().lower())
    relpaths = [p.relative_to(root) for p in files]

    tree = build_tree(relpaths)
    tree_lines = [root.name or root.as_posix()] + render_tree(tree)

    outpath.parent.mkdir(parents=True, exist_ok=True)

    now = _dt.datetime.now().isoformat(timespec="seconds")

    with outpath.open("w", encoding="utf-8") as out:
        # Header
        out.write("# Documentation Bundle\n\n")
        out.write(f"- **Root:** `{root}`\n")
        out.write(f"- **Generated:** `{now}`\n")
        out.write(f"- **Included files:** `{len(files)}`\n")
        out.write(f"- **Max text file bytes:** `{MAX_TEXT_FILE_BYTES}`\n")
        out.write(f"- **Ignored dirs:** `{''.join(sorted(IGNORE_DIRS))}`\n\n")

        # Tree view
        out.write("## Filesystem Tree (included paths)\n\n")
        out.write(fenced_block("\n".join(tree_lines), "text"))
        out.write("\n")

        # TOC
        out.write("## Table of Contents\n\n")
        for i, rp in enumerate(relpaths, start=1):
            out.write(f"{i}. [{rp.as_posix()}] (#file-{i})\n")
        out.write("\n")

        # File contents
```

```
out.write("## File Contents\n\n")
for i, path in enumerate(files, start=1):
    rp = path.relative_to(root).as_posix()
    out.write(f'<a id="file-{i}"></a>\n')
    out.write(f"### [{i}] `{rp}`\n\n")

    size = path.stat().st_size
    out.write(f"- **Bytes:** `{size}`\n")

    if is_png(path):
        out.write("- **Type:** `png`\n")
        dims = png_dimensions(path)
        if dims:
            out.write(f"- **Dimensions:** `{dims[0]}x{dims[1]}`\n")
        elif size == 0:
            out.write("- **Dimensions:** `unknown (empty file)`\n")
        else:
            out.write("- **Dimensions:** `unknown`\n")

    # Embed image (path relative to the markdown file location)
    md_rel = relpath_posix(outpath.parent, path)
    out.write(f"- **Path (from doc):** `{md_rel}`\n\n")
    out.write(f"![{rp}]({md_rel})\n\n")
    continue

# Text file
out.write("- **Type:** `text`\n")
content, note = read_text(path)
if note:
    out.write(f"- **NOTE:** {note}\n")
out.write("\n")

if content:
    lang = detect_code_lang(path)
    out.write(fenced_block(content, lang))
    out.write("\n")
else:
    out.write("_No content (skipped or empty)._\n\n")

return outpath

def main() -> int:
    parser = argparse.ArgumentParser(
        description="Bundle repository documentation into doc/doc.md (including .png images)."
    )
    parser.add_argument(
        "--root",
        type=Path,
        default=Path(__file__).resolve().parent,
        help="Root folder to scan (default: script directory).",
    )
    parser.add_argument(
```

```
    "--out",
    type=Path,
    default=None,
    help="Output markdown path (default: <root>/doc/doc.md).",
)
args = parser.parse_args()

root = args.root.resolve()
out = (args.out.resolve() if args.out else (root / "doc" / "doc.md"))

outpath = create_doc_md(root, out)
print(f"Documentation updated: Created '{outpath}'")
return 0

if __name__ == "__main__":
    raise SystemExit(main())
```

### [13] GenesisOS/apps.php

- **Bytes:** 16664
- **Type:** text

```
<?php
/**
 * Text Reversion
 * Applications Module
 *
 * This module manages the application catalogue, including listing built-in and
 * installed applications, and handling the submission/approval process for
 * new developer applications.
 */

/* =====
 * HELPER FUNCTIONS
 * ===== */
/** 
 * Loads the list of pending app submissions from its JSON file.
 * @return array A list of submission records.
 */
function loadAppSubmissions(): array {
    global $config;
    $file = $config['filesystem']['system_dir'] . '/app_submissions.json';
    if (!file_exists($file)) return [];
    $data = json_decode(file_get_contents($file), true);
    return is_array($data) ? $data : [];
}

/*
```

```
* Saves the list of pending app submissions to its JSON file.
* @param array $submissions The array of submissions to save.
*/
function saveAppSubmissions(array $submissions): void {
    global $config;
    $file = $config['filesystem']['system_dir'] . '/app_submissions.json';
    file_put_contents($file, json_encode($submissions, JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES));
}

/**
 * Validates the structure of an application manifest.
 * @param array $data The manifest data to validate.
 * @return bool True if the manifest is valid.
*/
function validateAppManifest(array $data): bool {
    $required = ['id', 'title', 'entry'];
    foreach ($required as $key) {
        if (empty($data[$key]) || !is_string($data[$key])) return false;
    }
    if (!preg_match('/^A-Za-z0-9_-]+$/i', $data['id'])) return false;
    if (isset($data['permissions']) && !is_array($data['permissions'])) return false;
    return true;
}

/**
 * Discovers application manifests from the filesystem.
 *  MODIFIED: Now adds a '_path' property to each manifest for correct frontend routing.
 * @return array A list of manifest arrays from installed applications.
*/
function discoverAppManifests(): array {
    global $config;
    static $cache = null;
    if ($cache !== null) return $cache;

    $list = [];
    $appDir = $config['filesystem']['apps_dir'];
    try {
        if (is_dir($appDir)) {
            foreach (new DirectoryIterator($appDir) as $dir) {
                if ($dir->isDot() || !$dir->isDir()) continue;

                //  MODIFIED: Find any .json file instead of the hardcoded 'app.json'.
                $manifests = glob($dir->getPathname() . '/*.json');
                if (empty($manifests)) continue; // Skip if no JSON file is found.

                $manifestPath = $manifests[0]; // Use the first JSON file found.

                $data = json_decode(file_get_contents($manifestPath), true);
                $appId = $dir->getBasename();
            }
        }
    } catch (Exception $e) {
        error_log("Error discovering app manifests: " . $e->getMessage());
    }
    return $list;
}
```

```

if (is_array($data) && validateAppManifest($data) && $data['id']
===$appId) {
    $data['_path'] = 'gos_files/apps/' . $appId . '/';
    $list[] = $data;
}
}

} catch (Throwable $e) {
    error_log('[discoverAppManifests] ' . $e);
}

return $cache = $list;
}

// In apps.php

/** 
 * Loads a single application manifest by its ID.
 *  MODIFIED: Now dynamically finds the .json file instead of assuming 'app.json'.
 * @param string $id The ID of the application.
 * @return array|null The manifest data or null if not found.
 */
function loadAppManifest(string $id): ?array {
    global $config;
    $appDir = $config['filesystem']['apps_dir'] . "/$id";

    // Find any .json file in the directory.
    $manifests = glob($appDir . '/*.json');
    if (empty($manifests)) {
        return null; // No manifest file found.
    }

    $file = $manifests[0]; // Use the first one found.

    if (!is_file($file)) return null;

    $json = json_decode(file_get_contents($file), true);
    return (is_array($json) && validateAppManifest($json)) ? $json : null;
}

/** 
 * Defines the list of applications that are built-in to the OS.
 * @return array An associative array of built-in app manifests.
 */
function builtinApps(): array {
    return [
        'desktop' => ['id' => 'desktop', 'title' => 'Desktop', 'description' => 'Genesis-OS desktop shell', 'icon' => '💻', 'category' => 'System', 'version' => '1.0.0', 'entry' => 'desktopApp', 'permissions' => [], 'window' => ['fullscreen' => true]],
        'mathematicalSandbox' => ['id' => 'mathematicalSandbox', 'title' => 'Mathematical Sandbox', 'description' => 'A grid-based environment for mathematical exploration', 'icon' => 'M²', 'category' => 'Science', 'version' => '1.0.0', 'entry' => 'mathematicalSandboxApp', 'permissions' => [], 'window' =>
    ]
}

```

```

['width' => 900, 'height' => 650, 'resizable' => true], 'help' => '<p>Create grids  
and evaluate expressions.</p>'],
    'fileManager' => ['id' => 'fileManager', 'title' => 'File Manager',  
'description' => 'Browse and manage files', 'icon' => '📁', 'category' =>  
'System', 'version' => '1.0.0', 'entry' => 'fileManagerApp', 'permissions' =>  
['filesystem.read.*', 'filesystem.write.*', 'process.exec'], 'window' => ['width'  
=> 900, 'height' => 650, 'resizable' => true], 'help' => '<p>Manage your files.  
</p>'],
    'terminal' => ['id' => 'terminal', 'title' => 'Terminal', 'description' =>  
'Command-line interface', 'icon' => '💻', 'category' => 'System', 'version' =>  
'1.0.0', 'entry' => 'terminalApp', 'permissions' => ['filesystem.read.*',  
'filesystem.write.*', 'process.exec'], 'window' => ['width' => 800, 'height' =>  
550, 'resizable' => true], 'help' => '<p>Type <code>help</code> for commands.  
</p>'],
    'editor' => ['id' => 'editor', 'title' => 'Code Editor', 'description' =>  
'Edit text and code files', 'icon' => '📝', 'category' => 'Development',  
'version' => '1.0.0', 'entry' => 'editorApp', 'permissions' =>  
['filesystem.read.*', 'filesystem.write.*'], 'window' => ['width' => 1000,  
'height' => 700, 'resizable' => true], 'help' => '<p>Use Ctrl+S to save.</p>'],
    'settings' => ['id' => 'settings', 'title' => 'Settings', 'description' =>  
'Configure system settings', 'icon' => '⚙️', 'category' => 'System', 'version' =>  
'1.0.0', 'entry' => 'settingsApp', 'permissions' => [], 'window' => ['width' =>  
600, 'height' => 500, 'resizable' => true], 'help' => '<p>Adjust system  
preferences.</p>'],
    'appStore' => ['id' => 'appStore', 'title' => 'App Store', 'description' =>  
'Install and manage applications', 'icon' => '📱', 'category' => 'System',  
'version' => '1.0.0', 'entry' => 'appStoreApp', 'permissions' => [], 'window' =>  
['width' => 1000, 'height' => 700, 'resizable' => true], 'help' => '<p>Browse and  
manage applications.</p>'],
    'diagnostics' => ['id' => 'diagnostics', 'title' => 'Diagnostics',  
'description' => 'Run system tests', 'icon' => '📝', 'category' => 'System',  
'version' => '1.0.0', 'entry' => 'diagnosticsApp', 'permissions' => [], 'window' =>  
['width' => 700, 'height' => 500, 'resizable' => true], 'adminOnly' => true,  
'help' => '<p>Admin tool to run tests.</p>'],
    'dev-center' => ['id' => 'dev-center', 'title' => 'Developer Center',  
'description' => 'Submit your applications.', 'icon' => '📌', 'category' =>  
'Development', 'version' => '1.0.0', 'entry' => 'devCenterApp', 'permissions' =>  
[], 'window' => ['width' => 800, 'height' => 700, 'resizable' => true],  
'developerOnly' => true, 'help' => '<p>Submit your apps for review.</p>']
];
}

/*
=====
* API-CALLABLE FUNCTIONS
=====
*/

```

---

```

function listApps(): array {
    $catalogue = [];
    $user = getCurrentUser();
    $isDeveloper = $user && in_array('developer', $user['roles'] ?? []);

```

---

```

    foreach (builtinApps() as $appId => $app) {
        if (($app['adminOnly'] ?? false) && !isAdmin()) continue;
        if (($app['developerOnly'] ?? false) && !$isDeveloper) continue;

```

```
$catalogue[$appId] = $app;
}

foreach (discoverAppManifests() as $app) {
    if (($app['adminOnly'] ?? false) && !isAdmin()) continue;
    if (($app['developerOnly'] ?? false) && !$isDeveloper) continue;
    $catalogue[$app['id']] = $app;
}

return array_values($catalogue);
}

/** 
 * Loads and returns the manifest for a single application by its ID.
 * This is called when an application is launched.
 *
 *  MODIFIED: This function now correctly adds the `'_path` property for
 * non-builtin apps, ensuring the frontend can resolve the entry point.
 *
 * @param array $params An array containing the 'id' of the app.
 * @return array|null The manifest data or null if not found.
 */
function getAppInfo(array $params): ?array {
    $id = $params['id'] ?? null;
    if (!$id) {
        return null;
    }

    // First, check if it's a built-in application.
    $builtInApps = builtinApps();
    if (isset($builtInApps[$id])) {
        return $builtInApps[$id];
    }

    // If not built-in, load it from the filesystem.
    $manifest = loadAppManifest($id);
    if ($manifest) {
        // This is the crucial step: add the web-accessible path.
        $manifest['_path'] = 'gos_files/apps/' . $id . '/';
        return $manifest;
    }

    // If not found anywhere, return null.
    return null;
}

function listSubmissions(): array {
    return loadAppSubmissions();
}

function submitApp(array $params): array {
    $currentUser = getCurrentUser();
    $manifestJson = $params['manifest'] ?? '';
    $code = $params['code'] ?? '';
}
```

```
if (empty($manifestJson) || empty($code)) {
    return ['success' => false, 'message' => 'Manifest and code cannot be
empty.'];
}
$manifest = json_decode($manifestJson, true);
if (json_last_error() !== JSON_ERROR_NONE) {
    return ['success' => false, 'message' => 'Invalid JSON in manifest.'];
}
if (!validateAppManifest($manifest)) {
    return ['success' => false, 'message' => 'Manifest validation failed.'];
}

$submissions = loadAppSubmissions();
$submissions[] = [
    'manifest' => $manifest,
    'code' => $code,
    'submitted_by' => $currentUser['username'],
    'submitted_at' => date('c')
];
saveAppSubmissions($submissions);

return ['success' => true, 'message' => 'Application submitted
successfully.'];
}

function approveSubmission(array $params): array {
    global $config;
    $id = $params['id'] ?? null;
    if (!$id) return ['success' => false, 'message' => 'Application ID not
provided.'];

    $submissions = loadAppSubmissions();
    foreach ($submissions as $i => $sub) {
        if (($sub['manifest']['id'] ?? '') === $id) {
            $dir = $config['filesystem']['apps_dir'] . "/$id";
            if (!is_dir($dir)) mkdir($dir, 0755, true);

            $scriptFilename = "$id.js";
            $manifestFilename = "$id.json";

            // ☑ CRITICAL FIX: Update the 'entry' field within the manifest data
            // itself.
            $sub['manifest']['entry'] = $scriptFilename;

            // Now, save the MODIFIED manifest data to the correctly named file.
            file_put_contents($dir . '/' . $manifestFilename,
                json_encode($sub['manifest'], JSON_PRETTY_PRINT));
            file_put_contents($dir . '/' . $scriptFilename, $sub['code'] ?? '');

            array_splice($submissions, $i, 1);
            saveAppSubmissions($submissions);
            return ['success' => true, 'message' => 'App approved'];
        }
    }
}
```

```
        }
        return ['success' => false, 'message' => 'Submission not found'];
    }

    function rejectSubmission(array $params): array {
        $id = $params['id'] ?? null;
        if (!$id) return ['success' => false, 'message' => 'Application ID not provided.'];

        $submissions = loadAppSubmissions();
        $initialCount = count($submissions);

        //  FIXED: Replaced arrow function (PHP 7.4+) with a standard anonymous function
        // for broader compatibility with PHP 7.0+ environments.
        $submissions = array_filter($submissions, function($sub) use ($id) {
            return ($sub['manifest']['id'] ?? '') !== $id;
        });

        if (count($submissions) < $initialCount) {
            saveAppSubmissions(array_values($submissions));
            return ['success' => true, 'message' => 'App rejected'];
        }
        return ['success' => false, 'message' => 'Submission not found'];
    }

    /**
     * Securely serves a requested asset (like app.js) for a specific application.
     * This prevents direct file access and handles security checks.
     */
    function getAppAsset(array $params): void {
        global $config;
        $appId = $params['appId'] ?? null;
        $file = $params['file'] ?? null;

        if (!$appId || !$file) {
            header("HTTP/1.1 400 Bad Request");
            echo /* Missing parameters */;
            exit;
        }

        if (preg_match('/^([A-Za-z0-9_-]+)$/', $appId) !== 1 || strpos($file, '..') !==
false || $file[0] === '/') {
            header("HTTP/1.1 403 Forbidden");
            echo /* Invalid path */;
            exit;
        }

        $assetPath = $config['filesystem']['apps_dir'] . "/$appId/$file";

        if (!is_file($assetPath)) {
            header("HTTP/1.1 404 Not Found");
            echo /* Asset not found */;
            exit;
        }
    }
}
```

```
}

//  FINAL FIX: Add headers to ensure a clean response and allow cross-origin access.
// This resolves the final browser security error.
if (ob_get_level()) {
    ob_end_clean(); // Clear any previous output buffers.
}
header("Access-Control-Allow-Origin: *"); // Allow the script to be loaded from any origin.
header('Content-Type: application/javascript; charset=utf-8');

readfile($assetPath);
exit;
}

/*
 * =====
 * API ROUTER AND HANDLER
 * ===== */
/***
 * Handles all incoming API requests for the applications module.
 * It routes requests to the appropriate function after checking permissions.
 */
function handleAppsAPI(): void {
    $method = $_POST['method'] ?? $_GET['method'] ?? 'listApps';

    //  FINAL FIX: Use $_GET for the asset request, and $_POST['params'] for all other API calls.
    // This allows the server to correctly handle both types of requests.
    if ($method === 'getAppAsset') {
        $params = $_GET;
    } else {
        $params = $_POST['params'] ?? [];
        if (is_string($params)) {
            $params = json_decode($params, true) ?? [];
        }
    }

    // The rest of the function's logic can now correctly dispatch the request.
    if ($method === 'getAppAsset') {
        getAppAsset($params); // This function handles its own exit.
        return;
    }

    jsonHeader();
    $response = null;

    try {
        switch ($method) {
            case 'listApps':
                $response = ['success' => true, 'data' => listApps()];
                break;
        }
    }
}
```

```

case 'getAppInfo':
    $data = getAppInfo($params);
    $response = $data
        ? [ 'success' => true, 'data' => $data]
        : [ 'success' => false, 'message' => 'App not found.' ];
    break;
case 'listSubmissions':
    if (!isAdmin()) throw new Exception('Access Denied');
    $response = [ 'success' => true, 'data' => listSubmissions()];
    break;
case 'submitApp':
    if (!hasPermission('app.submit')) throw new Exception('Access
Denied');
    $response = submitApp($params);
    break;
case 'approveSubmission':
    if (!isAdmin()) throw new Exception('Access Denied');
    $response = approveSubmission($params);
    break;
case 'rejectSubmission':
    if (!isAdmin()) throw new Exception('Access Denied');
    $response = rejectSubmission($params);
    break;
default:
    $response = [ 'success' => false, 'message' => "Unknown method:
{$method}"];
}
} catch (Throwable $e) {
    error_log("[AppsAPI] Exception in method '$method': " . $e->getMessage());
    $response = [ 'success' => false, 'message' => $e->getMessage()];
}

echo json_encode($response);
exit;
}

```

## [14] GenesisOS/auth.php

- **Bytes:** 10054
- **Type:** text

```

<?php
/**
 * Text Reversion
 * Authentication Module
 *
 * This module is the central authority for user authentication, session
management,
 * and permission definition. It handles login, logout, registration, and session
validation, and it establishes the permissions that other modules will enforce.
*/

```

```
/* =====
 * DATABASE HELPERS
 * ===== */
/***
 * Loads the user database from the users.json file.
 * @return array A list of user records.
 */
function loadUserDB(): array {
    global $config;
    $userDBFile = $config['filesystem']['system_dir'] . '/users.json';
    if (!file_exists($userDBFile)) {
        return [];
    }
    $data = json_decode(file_get_contents($userDBFile), true);
    return is_array($data) ? $data : [];
}

/***
 * Saves the user database to the users.json file.
 * @param array $users The array of user records to save.
 */
function saveUserDB(array $users): void {
    global $config;
    $userDBFile = $config['filesystem']['system_dir'] . '/users.json';
    file_put_contents($userDBFile, json_encode($users, JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES));
}

/* =====
 * SESSION & PERMISSION HELPERS
 * ===== */
/***
 * Checks if a user is currently authenticated.
 * @return bool True if a user is logged in, false otherwise.
 */
function isAuthenticated(): bool {
    return isset($_SESSION['user']);
}

/***
 * Retrieves the currently authenticated user's data from the session.
 * @return array|null The user's data array or null if not authenticated.
 */
function getCurrentUser(): ?array {
    return $_SESSION['user'] ?? null;
}

/***
 * Checks if the current user has the 'admin' role.
 * @return bool True if the user is an administrator.
 */

```

```
function isAdmin(): bool {
    $user = getCurrentUser();
    return $user && in_array('admin', $user['roles'] ?? [], true);
}

/**
 * Checks if the current user has a specific permission.
 * This is the central permission-checking function for the entire backend.
 * @param string $permission The permission string to check (e.g., 'filesystem.write.*').
 * @return bool True if the user has the permission.
 */
function hasPermission(string $permission): bool {
    if (!isAuthenticated()) {
        return false;
    }
    $permissions = $_SESSION['permissions'] ?? [];

    // Admins have all permissions implicitly.
    if (in_array('admin', $_SESSION['user']['roles'] ?? [])) {
        return true;
    }

    // Check for direct permission match.
    if (in_array($permission, $permissions, true)) {
        return true;
    }

    // Check for wildcard permissions (e.g., 'filesystem.write.*' grants 'filesystem.write.user').
    $parts = explode('.', $permission);
    while (count($parts) > 1) {
        array_pop($parts);
        $wildcard = implode('.', $parts) . '.*';
        if (in_array($wildcard, $permissions, true)) {
            return true;
        }
    }
}

return false;
}

/**
 * Defines the complete set of permissions for a given user based on their roles.
 * @param array $user The user record.
 * @return array A list of permission strings.
 */
function getUserPermissions(array $user): array {
    // Base permissions for all authenticated users.
    $permissions = [
        'app.launch.*',
        'filesystem.read.*',
        'filesystem.write.user.*',
        'filesystem.delete.user.*',
    ]
}
```

```
'sandbox.execute.user'
];

// Add administrator permissions.
if (in_array('admin', $user['roles'], true)) {
    $permissions = array_merge($permissions, [
        'filesystem.write.*',
        'filesystem.delete.*',
        'system.config.*',
        'system.admin.*',
        'sandbox.execute.*',
        'process.exec',
        'users.manage'
    ]);
}

// Add developer permissions.
if (in_array('developer', $user['roles'], true)) {
    $permissions = array_merge($permissions, [
        'app.submit',
        'filesystem.write.apps.self',
        'filesystem.delete.apps.self',
        'filesystem.write.*',
        'process.exec'
    ]);
}

return array_unique($permissions);
}

/*
 * =====
 * API HANDLER
 * ===== */
function handleAuthAPI(): array {
    global $config;

    $method = $_POST['method'] ?? $_GET['method'] ?? null;
    $params = $_POST['params'] ?? [];
    if (is_string($params)) {
        $params = json_decode($params, true) ?? [];
    }

    if ($method === null) {
        http_response_code(400);
        return ['success' => false, 'message' => 'Missing method'];
    }

    switch ($method) {
        case 'login':
            $username = $params['username'] ?? null;
```

```
$password = $params['password'] ?? null;
if (!$username || !$password) {
    return ['success' => false, 'message' => 'Missing username or password'];
}

$users = loadUserDB();
foreach ($users as $i => $record) {
    if ($record['username'] === $username &&
password_verify($password, $record['password'])) {
        $user = [
            'username' => $record['username'],
            'name' => $record['name'],
            'roles' => $record['roles'] ?? ['user'],
            'quota' => $record['quota'] ?? (10 * 1024 * 1024),
        ];
        $permissions = getUserPermissions($user);

        $_SESSION['user'] = $user;
        $_SESSION['permissions'] = $permissions;
        session_regenerate_id(true);

        $users[$i]['lastLogin'] = date('c');
        saveUserDB($users);
    }
    return ['success' => true, 'data' => ['user' => $user,
'permissions' => $permissions]];
}
}

return ['success' => false, 'message' => 'Invalid username or password'];

case 'register':
    if (!$config['security']['allow_developer_registration']) {
        return ['success' => false, 'message' => 'Developer registration is currently disabled.'];
    }
    $username = $params['username'] ?? null;
    $password = $params['password'] ?? null;
    $name = $params['name'] ?? null;

    if (!$username || !$password || !$name) {
        return ['success' => false, 'message' => 'Username, password, and name are required.'];
    }
    if (!preg_match('/^[\a-zA-Z0-9_]{3,20}$/', $username)) {
        return ['success' => false, 'message' => 'Username must be 3-20 characters and contain only letters, numbers, and underscores.'];
    }
    if (strlen($password) < 8) {
        return ['success' => false, 'message' => 'Password must be at least 8 characters long.'];
    }
```

```
$users = loadUserDB();
foreach ($users as $user) {
    if ($user['username'] === $username) {
        return ['success' => false, 'message' => 'Username is already
taken.'];
    }
}

$newUser = [
    'username' => $username,
    'password' => password_hash($password, PASSWORD_DEFAULT),
    'roles' => ['developer'],
    'name' => htmlspecialchars($name),
    'quota' => 15 * 1024 * 1024,
    'lastLogin' => null
];
$users[] = $newUser;
saveUserDB($users);

$userDir = $config['filesystem']['user_dir'] . '/' . $username;
if (!is_dir($userDir)) {
    mkdir($userDir . '/Desktop', 0755, true);
    mkdir($userDir . '/Documents', 0755, true);
}
return ['success' => true, 'message' => 'Registration successful. You
can now log in.'];

case 'logout':
// This provides a more robust logout suitable for HTTP and HTTPS.
// First, unset all session data.
$_SESSION = [];

// Next, delete the session cookie from the browser.
// This uses the session parameters (including the 'secure' flag set
in bootstrap.php)
// to ensure the cookie is properly targeted for deletion.
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"])
}
// Finally, destroy the session data on the server.
session_destroy();

return ['success' => true, 'message' => 'Logged out successfully'];

case 'validateToken':
$user = getCurrentUser();
if ($user) {
    $permissions = $_SESSION['permissions'] ??
getUserPermissions($user);
```

```

        $_SESSION['permissions'] = $permissions; // Refresh permissions in
session
            return ['success' => true, 'data' => ['valid' => true, 'user' =>
$user, 'permissions' => $permissions]];
        }
        return ['success' => true, 'data' => ['valid' => false]];

    default:
        return ['success' => false, 'message' => 'Invalid authentication
method'];
    }
}

```

## [15] GenesisOS/bootstrap.php

- **Bytes:** 3325
- **Type:** text

```

<?php
/**
 * Text Reversion
 * Bootstrapping File
 */

// Start output buffering to prevent header issues
ob_start();

// Load Configuration
$config = require_once 'config.php';

/*
-----
 * REQUIRED: Accept JSON bodies as if they were normal $_POST data
 * -----
 */
//if ($_SERVER['REQUEST_METHOD'] === 'POST' && !empty($_SERVER['CONTENT_TYPE']) &&
strpos($_SERVER['CONTENT_TYPE'], 'application/json') === 0) {
if (($_SERVER['REQUEST_METHOD'] ?? 'GET') === 'POST' &&
!empty($_SERVER['CONTENT_TYPE']) && strpos($_SERVER['CONTENT_TYPE'],
'application/json') === 0) {
    $raw = file_get_contents('php://input');
    if ($raw !== false && $raw !== '') {
        $json = json_decode($raw, true);
        if (json_last_error() === JSON_ERROR_NONE && is_array($json)) {
            // Merge so that explicit form fields can still override
            $_POST = array_merge($json, $_POST);
        }
    }
}

/*
-----
 * REQUIRED: Robust Error & Exception Handling
 * -----
 */

```

```
$errorLogFile = $config['filesystem']['system_dir'] . '/error.log';
ini_set('log_errors', '1');
ini_set('error_log', $errorLogFile);

set_error_handler(function($severity, $message, $file, $line) {
    $entry = '[' . date('Y-m-d H:i:s') . "] PHP [$severity] $message in $file:$line";
    error_log($entry);
    return false; // Allow default PHP handler to run as well
});

set_exception_handler(function($e) use ($errorLogFile) {
    $entry = '[' . date('Y-m-d H:i:s') . "] Uncaught Exception: {$e->getMessage()} in
{$e->getFile()}:{${e->getLine()}}\n".$e->getTraceAsString();
    error_log($entry);
});

register_shutdown_function(function() use ($errorLogFile) {
    $err = error_get_last();
    if ($err && in_array($err['type'], [E_ERROR, E_PARSE, E_CORE_ERROR,
E_COMPILE_ERROR])) {
        $entry = '[' . date('Y-m-d H:i:s') . "] Fatal Error: {$err['message']} in
{$err['file']}:{${err['line']}}";
        file_put_contents($errorLogFile, $entry . "\n", FILE_APPEND);
    }
});
/* -----
 * Session and Filesystem Initialization
 * ----- */
$secureCookie = (!empty($_SERVER['HTTPS']) && $_SERVER['HTTPS'] != 'off');
session_set_cookie_params([
    'lifetime' => 0,
    'path' => '/',
    'secure' => $secureCookie,
    'httponly' => true,
    'samesite' => 'Lax'
]);
session_start();

// Include Core Modules
require_once 'modules/auth.php';
require_once 'modules/filesystem.php';
require_once 'modules/apps.php';
require_once 'modules/users.php';
require_once 'modules/system.php';
require_once 'modules/sandbox.php';

// Create required directories if filesystem is enabled
if ($config['filesystem']['use_local']) {
    foreach ($config['filesystem'] as $key => $dir) {
        // Use is_dir for a more reliable check
        if (strpos($key, 'dir') !== false && !is_dir($dir)) {
            mkdir($dir, 0755, true);
        }
    }
}
```

```
        }
    }

// Helper for consistent JSON headers, to be used in index.php
function jsonHeader(): void {
    if (!headers_sent()) {
        header('Content-Type: application/json; charset=utf-8');
    }
}
```

## [16] GenesisOS/charset-grid-viewer.js

- **Bytes:** 33320
- **Type:** text

```
export function initialize(gosApiProxy) {
    const container = gosApiProxy.window.getContainer();
    const doc = container.ownerDocument;

    // --- 1. Inject CSS Styles ---
    const style = doc.createElement('style');
    style.textContent = `
        :root {
            --primary-color: #4B5320; --secondary-color: #FFFFFF; --accent-color: #bdb76b;
            --background-color: #f5f5dc; --text-color: #333333; --font-main: "Fira Code", "Consolas", monospace;
        }
        body { font-family: var(--font-main); background-color: var(--background-color); color: var(--text-color); margin: 0; padding: 20px; box-sizing: border-box; }
        header { text-align: center; margin-bottom: 20px; padding-bottom: 20px; border-bottom: 4px double var(--primary-color); }
        main { max-width: 1400px; margin: auto; }
        h2 { color: var(--primary-color); text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.1); font-weight: 600; margin-top: 0; }
        h3 { color: var(--primary-color); }

        /* Tab Styles */
        .app-tabs { display: flex; border-bottom: 2px solid var(--primary-color); margin-bottom: 20px; }
        .tab-button { background-color: transparent; color: var(--text-color); border: none; padding: 12px 18px; cursor: pointer; font-size: 1.1em; transition: background-color 0.3s, color 0.3s; border-radius: 5px 5px 0 0; margin: 0 5px -2px 5px; border: 2px solid transparent; }
        .tab-button:hover { background-color: rgba(75, 83, 32, 0.1); }
        .tab-button.active { background-color: var(--secondary-color); border-color: var(--primary-color) var(--primary-color) var(--secondary-color) var(--primary-color); font-weight: bold; }
        .content-panel { display: none; }
        .content-panel.active { display: block; animation: fadeIn 0.5s; }
```

```
@keyframes fadeIn { from { opacity: 0; } to { opacity: 1; } }

/* Original App Styles */
.operation-section { background-color: var(--secondary-color); padding: 20px; border: 2px solid var(--accent-color); box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); margin-bottom: 20px; }
.operation-container { display: flex; flex-wrap: wrap; gap: 20px; }
.operation-controls, .operation-result { flex: 1; min-width: 300px; }
button { background-color: var(--primary-color); color: var(--secondary-color); padding: 10px 15px; font-size: 1em; border: 2px solid var(--accent-color); cursor: pointer; transition: all 0.3s ease; font-family: var(--font-main); margin-top: 5px; }
button:hover { background-color: var(--accent-color); color: var(--text-color); transform: translateY(-2px); }
button:disabled { background-color: #ccc; border-color: #999; color: #666; cursor: not-allowed; transform: none; }
input, select, textarea { width: 100%; padding: 10px; font-size: 1em; background-color: #fefefe; color: var(--text-color); border: 2px solid var(--accent-color); font-family: var(--font-main); margin-bottom: 10px; box-sizing: border-box; }
textarea { resize: vertical; }
.result-container { background-color: #FFFFFF; padding: 10px; border: 2px solid var(--accent-color); font-family: var(--font-main); box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1) inset; color: var(--text-color); word-wrap: break-word; white-space: pre-wrap; margin-top: 10px; min-height: 50px; }
#map-system-section #option-container { margin-top: 15px; border-top: 2px solid var(--background-color); padding-top: 15px; }
#map-system-section #main-menu { display: flex; flex-wrap: wrap; gap: 10px; }
#color-bar { width: 100%; height: 20px; background-color: var(--accent-color); margin-top: 15px; border: 2px solid var(--primary-color); box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2); transition: background-color 0.3s ease; }
#custom-string { white-space: pre; overflow-wrap: normal; overflow-x: auto; }
#grid-canvas { border: 2px solid var(--accent-color); margin-top: 10px; width: 100%; max-width: 400px; height: auto; }
.navigation-controls { display: flex; justify-content: center; align-items: center; gap: 10px; width: 100%; margin-top: 10px; flex-wrap: wrap; }
.color-list-container { margin-top: 15px; }
.color-list { list-style-type: none; padding: 0; margin: 0; max-height: 200px; overflow-y: auto; border: 1px solid var(--accent-color); }
.color-list li { background: var(--background-color); padding: 5px 10px; border-bottom: 1px solid var(--accent-color); }
`;

doc.head.appendChild(style);

// --- 2. Inject Tabbed HTML Structure ---
container.innerHTML = `
<header>
    <h1>Integrated Charset & Grid Viewer</h1>
    <p>A tool for converting strings to unique IDs and visualizing them as tile grids.</p>
</header>
<div class="app-tabs">
```

```
<button class="tab-button active" data-tab="id-generation">ID  
Generation</button>  
    <button class="tab-button" data-tab="grid-visualization">Grid  
Visualization</button>  
</div>  
<main class="app-content-panels">  
    <div id="panel-id-generation" class="content-panel active">  
        <section id="charset-config-section" class="operation-section">  
            <h2>Charset Configuration</h2>  
            <p>Select the character sets to include in the alphabet.  
Changes apply to the current session only.</p>  
            <div id="charset-toggles-container" style="display: grid;  
grid-template-columns: repeat(auto-fill, minmax(250px, 1fr)); gap: 10px; margin-  
top: 10px; max-height: 150px; overflow-y: auto; border: 1px solid var(--accent-  
color); padding: 10px;">  
                </div>  
            <h3 style="margin-top: 20px;">Custom Characters</h3>  
            <p>Add any additional unique characters to the charset.</p>  
            <textarea id="custom-charset-input" rows="3"  
placeholder="e.g., ©™®"></textarea>  
            <button id="update-charset-btn" style="margin-top:  
10px;">Update Charset</button>  
        </section>  
        <section id="map-system-section" class="operation-section">  
            <div class="operation-container">  
                <div class="operation-controls">  
                    <h2>ID Generation Controls</h2>  
                    <div id="main-menu">  
                        <button id="show-generate-combinations">Generate  
Combinations</button>  
                        <button id="show-calculate-string-id">Calculate  
String ID</button>  
                        <button id="show-decode-id">Decode ID to  
String</button>  
                        <button id="show-find-optimal-variable">Find  
Optimal Variable</button>  
                    </div>  
                    <div id="option-container" style="display: none;">  
                        <div id="generate-combinations" style="display:  
none;">  
                            <h3>Generate Combinations</h3>  
                            <label for="combination-size">Enter the size  
of combinations (n):</label>  
                            <input type="number" id="combination-size">  
                            <label for="output-file-generate">Enter the  
name of the output file:</label>  
                            <input type="text" id="output-file-generate"  
value="combinations.txt">  
                            <button id="generate-combinations-  
btn">Generate</button>  
                        </div>  
                        <div id="calculate-string-id" style="display:  
none;">  
                            <h3>Calculate String ID</h3>  
                        </div>  
                    </div>  
                </div>  
            </div>  
        </section>  
    </div>  
</main>
```

```
        <label for="custom-string">Enter your custom
string:</label>
<cols="50" wrap="off"></textarea>
<button id="calculate-string-id-
btn">Calculate</button>
<container"></p>
Words: 0</div>
</label>
<div id="decode-id" style="display: none;">
<h3>Decode ID to String</h3>
<label for="string-id">Enter the ID to decode:
<input type="text" id="string-id">
<button id="decode-id-btn">Decode</button>
<p id="decoded-string-result" class="result-
container"></p>
</div>
<div id="find-optimal-variable" style="display:
none;">
<h3>Find Optimal Variable and Save to
File</h3>
<label for="user-number">Enter the user
number:</label>
<input type="number" id="user-number">
<label for="output-file-optimal">Enter the
name of the output file:</label>
<input type="text" id="output-file-optimal"
value="optimal_variables.txt">
<button id="find-optimal-variable-btn">Find
and Save</button>
</div>
</div>
</div>
<div class="operation-result">
<h3>Result</h3>
<div id="map-result-display" class="result-container">
<p>Select an operation to start.</p>
<h3>Grid Validation Status</h3>
<div id="color-bar"></div>
</div>
</div>
</section>
</div>
<div id="panel-grid-visualization" class="content-panel">
<section id="alphabet-system-section" class="operation-section">
<div class="operation-container">
<div class="operation-controls">
<h2>Grid Generation Controls</h2>
<div id="controls">
<label for="tile-size">Tile Size (px):</label>
<input type="number" id="tile-size" value="100">
```

```
</div>
<div id="compounded-id-input-container">
    <h3>Generate Grid by Compounded ID</h3>
    <p>Enter an ID from the Charset App or upload a
file with IDs.</p>
    <div id="compounded-id-inputs">
        <div class="compounded-id-input-row">
            <input type="text" class="compounded-id-
input" placeholder="Enter Compounded Grid ID">
        </div>
    </div>
    <button id="generate-grid-by-id">Generate
Grids</button>
    <hr style="margin: 15px 0;">
    <input type="file" id="file-input" accept=".txt">
    <button id="execute-button">Process File</button>
</div>
</div>
<div class="operation-result">
    <h3>Grid Display</h3>
    <canvas id="grid-canvas" style="background-color:
#eee;"></canvas>
    <div id="navigation-controls">
        <button id="prev-grid" disabled>Previous
Grid</button>
        <span id="current-grid-position">Grid 0 of
0</span>
        <button id="next-grid" disabled>Next Grid</button>
    </div>
    <button id="download-grid-image" disabled>Download
Grid Image</button>
    <div id="color-list-container">
        <h3>Current Grid Tile IDs</h3>
        <div id="current-grid-id"></div>
        <ul id="color-list"></ul>
    </div>
</div>
</div>
</section>
</div>
</main>
`;

// --- 3. Application Logic ---

const tabs = doc.querySelectorAll('.tab-button');
const panels = doc.querySelectorAll('.content-panel');
const switchTab = (targetTabId) => {
    panels.forEach(panel => panel.classList.remove('active'));
    doc.getElementById(`panel-${targetTabId}`).classList.add('active');
    tabs.forEach(tab => tab.classList.remove('active'));
    doc.querySelector(`.tab-button[data-
tab="${targetTabId}"]`).classList.add('active');
};
```

```
tabs.forEach(tab => tab.addEventListener('click', () =>
switchTab(tab.dataset.tab));

const mapApp = (() => {
  let _charsetConfig = [], _uniqueCharset = [];

  function initialize() {
    _initCharsetConfig();
    _populateCharsetToggles();
    _updateAndInitializeCharset();
    _initEventListeners();
  }

  function _initCharsetConfig() {
    _charsetConfig = [
      { name: 'Basic Latin', range: [0x0020, 0x007F] }, { name: 'Latin-1 Supplement', range: [0x0080, 0x00FF] },
      { name: 'Latin Extended-A', range: [0x0100, 0x017F] }, { name: 'Latin Extended-B', range: [0x0180, 0x024F] },
      { name: 'Greek and Coptic', range: [0x0370, 0x03FF] }, { name: 'Cyrillic', range: [0x0400, 0x04FF] },
      { name: 'Arabic', range: [0x0600, 0x06FF] }, { name: 'Hebrew', range: [0x0590, 0x05FF] },
      { name: 'Devanagari', range: [0x0900, 0x097F] }, { name: 'Mathematical Operators', range: [0x2200, 0x22FF] },
      { name: 'Supplemental Mathematical Operators', range: [0x2A00, 0x2AFF] }, { name: 'Miscellaneous Technical', range: [0x2300, 0x23FF] },
      { name: 'Miscellaneous Symbols and Arrows', range: [0x2190, 0x21FF] }, { name: 'CJK Unified Ideographs', range: [0x4E00, 0x9FFF] },
      { name: 'Hangul Syllables', range: [0xAC00, 0xD7AF] }, { name: 'Hiragana', range: [0x3040, 0x309F] },
      { name: 'Katakana', range: [0x30A0, 0x30FF] }, { name: 'Bopomofo', range: [0x3100, 0x312F] },
      { name: 'Currency Symbols', range: [0x20A0, 0x20CF] }, { name: 'Additional Punctuation', range: [0x2000, 0x206F] }
    ];
  }

  function _populateCharsetToggles() {
    const container = doc.getElementById('charset-toggles-container');
    if (!container) return;
    container.innerHTML = _charsetConfig.map((block, index) =>
      <label style="display: block; white-space: nowrap; overflow: hidden; text-overflow: ellipsis;" title="${block.name}">
        <input type="checkbox" class="charset-toggle" data-index="${index}" checked>
          ${block.name}
      </label>
    ).join('');
  }

  function _updateAndInitializeCharset() {
    const resultDisplay = doc.getElementById('map-result-display');
    try {

```

```
        const activeToggles = doc.querySelectorAll('.charset-
toggle:checked');
        const activeConfig = Array.from(activeToggles).map(toggle =>
_charsetConfig[parseInt(toggle.dataset.index)]);
        const customCharsInput = doc.getElementById('custom-charset-
input').value;

        const charset = new Set();
        activeConfig.forEach(block => {
            for (let i = block.range[0]; i <= block.range[1]; i++) {
                try { charset.add(String.fromCharCode(i)); } catch (e) {}
            }
        });
        charset.add('\n'); charset.add('\t');
        for (const char of customCharsInput) { charset.add(char); }

        _uniqueCharset = Array.from(charset);
        const message = `Charset updated with ${_uniqueCharset.length}
characters.`;
        gosApiProxy.ui.showNotification('Charset Updated', message, 3000);
        if(resultDisplay) resultDisplay.innerHTML = `<p class="success-
message">${message}</p>`;
    } catch (error) {
        const message = `Failed to update charset: ${error.message}`;
        gosApiProxy.ui.showNotification('Error', message, 4000);
        if(resultDisplay) resultDisplay.innerHTML = `<p class="error-
message">${message}</p>`;
    }
}

function _initEventListeners() {
    doc.getElementById('update-charset-btn').addEventListener('click',
_updateAndInitializeCharset);
    doc.getElementById('show-generate-
combinations').addEventListener('click', () => showOption('generate-
combinations'));
    doc.getElementById('show-calculate-string-
id').addEventListener('click', () => showOption('calculate-string-id'));
    doc.getElementById('show-decode-id').addEventListener('click', () =>
showOption('decode-id'));
    doc.getElementById('show-find-optimal-
variable').addEventListener('click', () => showOption('find-optimal-variable'));
    doc.getElementById('generate-combinations-
btn').addEventListener('click', generateCombinations);
    doc.getElementById('calculate-string-id-
btn').addEventListener('click', calculateStringID);
    doc.getElementById('decode-id-btn').addEventListener('click',
decodeID);
    doc.getElementById('find-optimal-variable-
btn').addEventListener('click', findOptimalVariable);
    doc.getElementById('custom-string').addEventListener('input', (e) =>
displayCharacterAndWordCount(e.target.value));
}
```

```
function showOption(optionId) {
    doc.querySelectorAll('#option-container > div').forEach(opt =>
    opt.style.display = 'none');
    doc.getElementById(optionId).style.display = 'block';
    doc.getElementById('option-container').style.display = 'block';
}

function downloadFile(filename, content) {
    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = window.parent.document.createElement('a'); // ✅ FIX:
Create element in parent document for download
    a.href = url; a.download = filename;
    window.parent.document.body.appendChild(a);
    a.click();
    window.parent.document.body.removeChild(a);
    URL.revokeObjectURL(url);
}

function calculateStringID() {
    const resultDisplay = doc.getElementById('map-result-display');
    try {
        const inputString = doc.getElementById('custom-string').value; if
(inputString.trim() === '') throw new Error("Input string cannot be empty.");
        resultDisplay.innerHTML = '<p>Calculating...</p>';
        setTimeout(() => {
            try {
                const invalidChars = [...new
Set([...inputString].filter(char => !_uniqueCharset.includes(char))]];
                if (invalidChars.length > 0) throw new Error(`Invalid
characters found (not in current charset): ${invalidChars.join('')}`);
                let id = 0n;
                for (const char of inputString) { id = id *
BigInt(_uniqueCharset.length) + BigInt(_uniqueCharset.indexOf(char)); }
                const resultText = id.toString();
                doc.getElementById('string-id-result').innerText = "Result
ID: " + resultText;
                const colorIndexes = decodeIDtoColorIndexes(resultText);
                updateColorBar(colorIndexes);
                resultDisplay.innerHTML = `<p> Calculation complete. ID:
<strong>${resultText}</strong></p><button id="send-id-to-viewer">Visualize This
ID</button>`;
                doc.getElementById('send-id-to-
viewer').addEventListener('click', () => {
                    alphabetApp.setInputIDAndGenerate(resultText);
                    switchTab('grid-visualization');
                });
                } catch (error) { resultDisplay.innerHTML = `<p
style="color:red;">${error.message}</p>`;
}, 50);
            } catch (error) { resultDisplay.innerHTML = `<p
style="color:red;">${error.message}</p>`;
}
    }
}
```

```
function decodeIDtoColorIndexes(idString) { if (!idString) return [];
return (idString.match(/.{1,7}/g) || []).map(s => parseInt(s, 10)); }

function updateColorBar(indexes) {
    const colorBar = doc.getElementById('color-bar'); const isSq = (n) =>
n > 0 && Math.sqrt(n) % 1 === 0;
    const isValid = (idxs) => idxs.every(index => index >= 1 && index <=
16777216);
    if (isSq(indexes.length) && isValid(indexes)) {
        colorBar.style.backgroundColor = 'green'; colorBar.title = "Valid:
The number of color indexes forms a perfect square.";
    } else { colorBar.style.backgroundColor = 'red'; colorBar.title =
"Invalid: The number of color indexes does not form a perfect square for a grid.";
}
}

function displayCharacterAndWordCount(inputText) {
    const charCount = inputText.length; const wordCount = inputText.trim() === '' ? 0 : inputText.trim().split(/\s+/).length;
    doc.getElementById('character-word-count').innerText = `Characters:
${charCount}, Words: ${wordCount}`;
}

function generateCombinations() {
    const resultDisplay = doc.getElementById('map-result-display');
    try {
        const n = BigInt(doc.getElementById('combination-size').value);
        const outputFile = doc.getElementById('output-file-
generate').value || 'combinations.txt';
        if (n <= 0n) throw new Error("Combination size must be a positive
integer.");
        if (n > 4n && !confirm(`This may generate a very large number of
combinations. Continue?`)) return;

        resultDisplay.innerHTML = '<p>Generating combinations... This may
take a while.</p>';
        setTimeout(() => {
            try {
                const k = BigInt(_uniqueCharset.length);
                const nbrComb = k ** n;
                const maxToGenerate = 1000000n;
                const actualGenerate = nbrComb > maxToGenerate ?
maxToGenerate : nbrComb;
                let combinations = '';
                for (let i = 0n; i < actualGenerate; i++) {
                    let id = i; let combination = '';
                    for (let j = 0n; j < n; j++) {
                        combination = _uniqueCharset[Number(id % k)] +
combination;
                        id /= k;
                    }
                    combinations += combination + '\n';
                }
                downloadFile(outputFile, combinations);
            }
        });
    }
}
```

```
        resultDisplay.innerHTML = `<p>Successfully generated
${actualGenerate.toLocaleString()} combinations and saved to ${outputFile}.</p>`;
    } catch (error) { resultDisplay.innerHTML = `<p
style="color:red;">Error during generation: ${error.message}</p>`; }
}, 50);
} catch (error) { resultDisplay.innerHTML = `<p
style="color:red;">${error.message}</p>`; }
}

function decodeID() {
    const resultDisplay = doc.getElementById('map-result-display');
    try {
        const idStr = doc.getElementById('string-id').value;
        if (!/^\\d+$/.test(idStr)) throw new Error("ID must be a valid non-
negative integer.");
        let id = BigInt(idStr);
        resultDisplay.innerHTML = '<p>Decoding ID...</p>';
        setTimeout(() => {
            try {
                let decodedString = [];
                if (id === 0n && _uniqueCharset.length > 0) {
                    decodedString.push(_uniqueCharset[0]);
                } else {
                    while (id > 0n) {
                        decodedString.push(_uniqueCharset[Number(id %
BigInt(_uniqueCharset.length))]);
                        id /= BigInt(_uniqueCharset.length);
                    }
                }
                const decodedResult = decodedString.reverse().join('');
                doc.getElementById('decoded-string-result').innerText =
"Decoded String: " + decodedResult;
                resultDisplay.innerHTML = `<p>Decoding complete. Result:
<pre>${decodedResult}</pre></p>`;
            } catch (error) { resultDisplay.innerHTML = `<p
style="color:red;">Error decoding ID: ${error.message}</p>`; }
}, 50);
} catch (error) { resultDisplay.innerHTML = `<p
style="color:red;">${error.message}</p>`; }
}

function findOptimalVariable() {
    const resultDisplay = doc.getElementById('map-result-display');
    try {
        const userNumber = BigInt(doc.getElementById('user-
number').value);
        const outputFile = doc.getElementById('output-file-optimal').value
|| 'optimal_variables.txt';
        if (userNumber <= 0n) throw new Error("User number must be
positive.");
        resultDisplay.innerHTML = '<p>Finding optimal variables...</p>';
        setTimeout(() => {
            try {
                const charsetLength = BigInt(_uniqueCharset.length);

```

```
        const generator = (function*( u) { for (let k = 1n; ; k++)  
{ yield k * u; } })(userNumber);  
        const decode = (num) => {  
            let decStr = []; if (num === 0n) return  
_uniqueCharset[0] || '';  
            while(num > 0n){ decStr.push(_uniqueCharset[Number(num  
% charsetLength)]); num /= charsetLength; }  
            return decStr.reverse().join('');  
};  
        const results = Array.from({length: 100}, () => { const  
optVar = generator.next().value; const decodedString = decode(optVar); return {  
variable: optVar, string: decodedString, length: decodedString.length }; });  
        const resultString = results.map(r =>  
` ${r.variable}\t${r.string}`).join('\n');  
        downloadFile(outputFile, resultString);  
        let resultsHTML = `<p>Found 100 optimal variables. File  
'${outputFile}' downloaded.</p><p>Showing first 10:</p><table  
style="width:100%;text-align:left;"><tr><th>Variable</th><th>String</th>  
<th>Length</th></tr>`;  
        results.slice(0, 10).forEach(r => { resultsHTML += `<tr>  
<td>${r.variable}</td><td>${r.string}</td><td>${r.length}</td></tr>` );  
        resultsHTML += `</table>`;  
        resultDisplay.innerHTML = resultsHTML;  
    } catch (error) { resultDisplay.innerHTML = `<p  
style="color:red;">Error finding optimal variables: ${error.message}</p>` ; }  
, 50);  
    } catch (error) { resultDisplay.innerHTML = `<p  
style="color:red;">${error.message}</p>` ; }  
}  
    return { initialize };  
})();  
  
const alphabetApp = (() => {  
    let _grids = [], _currentGridIndex = 0, _defaultTileSize = 100;  
    function initialize() { _initEventListeners(); displayCurrentGrid(); }  
    function _initEventListeners() {  
        doc.getElementById('tile-size').addEventListener('input', () =>  
displayCurrentGrid());  
        doc.getElementById('generate-grid-by-id').addEventListener('click', generateGridsFromInputs);  
        doc.getElementById('execute-button').addEventListener('click', processUploadedFile);  
        doc.getElementById('prev-grid').addEventListener('click', () =>  
navigateGrid(-1));  
        doc.getElementById('next-grid').addEventListener('click', () =>  
navigateGrid(1));  
        doc.getElementById('download-grid-image').addEventListener('click', downloadGridImage);  
    }  
    function setInputIDAndGenerate(id) {  
        const inputField = doc.querySelector('.compounded-id-input');  
inputField.value = id;  
        generateGridsFromInputs();  
    }  
});
```

```
function idToHex(id) { id = BigInt(id); if (id <= 0n) return '#000000';  
return `#${(id - 1n).toString(16).padStart(6, '0')}`; }  
function calculateGridDimensions(numTiles) {  
    if (numTiles === 0) return { rows: 0, cols: 0 }; const sideLength =  
Math.ceil(Math.sqrt(numTiles));  
    return { rows: sideLength, cols: sideLength };  
}  
function displayCurrentGrid() {  
    const canvas = doc.getElementById('grid-canvas'); const context =  
canvas.getContext('2d');  
    const colorSequence = _grids[_currentGridIndex] || []; const numTiles  
= colorSequence.length;  
    if (numTiles === 0) {  
        canvas.width = 400; canvas.height = 100; context.clearRect(0, 0,  
canvas.width, canvas.height);  
        context.font = "16px " + getComputedStyle(doc.body).fontFamily;  
context.fillStyle = "#888";  
        context.textAlign = "center"; context.fillText("No grid to  
display.", canvas.width / 2, canvas.height / 2);  
        updateUIElements(); return;  
    }  
    const tileSize = parseInt(doc.getElementById('tile-size').value) ||  
_defaultTileSize;  
    const { rows, cols } = calculateGridDimensions(numTiles);  
    canvas.width = cols * tileSize; canvas.height = rows * tileSize;  
    context.clearRect(0, 0, canvas.width, canvas.height);  
    colorSequence.forEach((color, i) => {  
        const col = i % cols, row = Math.floor(i / cols);  
        context.fillStyle = color;  
        context.fillRect(col * tileSize, row * tileSize, tileSize,  
tileSize);  
        context.strokeStyle = '#ccc'; context.strokeRect(col * tileSize,  
row * tileSize, tileSize, tileSize);  
    });  
    updateUIElements();  
}  
function updateUIElements() {  
    const colorSequence = _grids[_currentGridIndex] || [];  
    doc.getElementById('color-list').innerHTML = colorSequence.map(color  
=> `<li>ID ${BigInt('0x' + color.replace('#', '')) + 1n}: ${color}  
/</li>`).join('');  
    doc.getElementById('current-grid-position').textContent = `Grid  
${_grids.length} > 0 ? _currentGridIndex + 1 : 0} of ${_grids.length}`;  
    doc.getElementById('prev-grid').disabled = _currentGridIndex === 0;  
    doc.getElementById('next-grid').disabled = _currentGridIndex >=  
_grids.length - 1;  
    doc.getElementById('download-grid-image').disabled =  
colorSequence.length === 0;  
}  
function navigateGrid(direction) {  
    const newIndex = _currentGridIndex + direction;  
    if (newIndex >= 0 && newIndex < _grids.length) { _currentGridIndex =  
newIndex; displayCurrentGrid(); }  
}
```

```

        function regenerateGrids(compoundedIDs) {
            _grids = compoundedIDs.map(idString => (idString.trim().match(/.
{1,7}/g) || []).map(id => idToHex(id)));
            _currentGridIndex = 0; displayCurrentGrid();
            gosApiProxy.ui.showNotification("Success", `${_grids.length} grid(s)
regenerated successfully.`);
        }
        function generateGridsFromInputs() {
            const ids = [...doc.querySelectorAll('.compounded-id-input')].map(i =>
i.value.trim()).filter(id => id);
            if (ids.length === 0) { gosApiProxy.ui.showNotification("Error",
'Please enter at least one ID.', 4000); return; }
            regenerateGrids(ids);
        }
        function processUploadedFile() {
            const file = doc.getElementById('file-input').files[0];
            if (!file) { gosApiProxy.ui.showNotification("Error", "Please select a
file.", 4000); return; }
            const reader = new FileReader();
            reader.onload = (e) => {
                const ids = e.target.result.split('\n').map(l =>
l.trim()).filter(l => /^d+$/.test(l));
                if (ids.length > 0) regenerateGrids(ids);
                else gosApiProxy.ui.showNotification("Error", "No valid IDs found
in file.", 4000);
            };
            reader.readAsText(file);
        }
        function downloadGridImage() {
            const canvas = doc.getElementById('grid-canvas'); const image =
canvas.toDataURL('image/png');
            const a = window.parent.document.createElement('a'); // ✅ FIX:
Create element in parent document for download
            a.href = image; a.download = `grid_${_currentGridIndex + 1}.png`;
            window.parent.document.body.appendChild(a);
            a.click();
            window.parent.document.body.removeChild(a);
            URL.revokeObjectURL(a.href);
        }
        return { initialize, setInputIDAndGenerate };
    })();

    mapApp.initialize();
    alphabetApp.initialize();
}

```

## [17] GenesisOS/charset-grid-viewer.json

- **Bytes:** 491
- **Type:** text

```
{  
    "id": "charset-grid-viewer",  
    "title": "Charset Grid Viewer",  
    "description": "An advanced tool to generate unique IDs from text using a  
custom charset and visualize those IDs as colored tile grids.",  
    "version": "1.0.0",  
    "author": "GOS Conversion",  
    "icon": "\ud83c\udc04\ufe0f",  
    "category": "Development",  
    "entry": "charset-grid-viewer.js",  
    "permissions": [],  
    "window": {  
        "width": 1450,  
        "height": 800,  
        "resizable": true  
    }  
}
```

## [18] GenesisOS/clix-repl.html

- **Bytes:** 57787
- **Type:** text

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>CLIX REPL System</title>  
    <style>  
        * {  
            margin: 0;  
            padding: 0;  
            box-sizing: border-box;  
        }  
  
        body {  
            background: #0a0a0a;  
            color: #00ff00;  
            font-family: 'Consolas', 'Courier New', monospace;  
            height: 100vh;  
            display: flex;  
            flex-direction: column;  
            overflow: hidden;  
        }  
  
        #terminal {  
            flex: 1;  
            padding: 20px;  
            overflow-y: auto;  
        }</style>
```

```
        background: #000;
        position: relative;
    }

.terminal-line {
    margin-bottom: 2px;
    white-space: pre-wrap;
    word-wrap: break-word;
    line-height: 1.4;
}

.terminal-prompt {
    color: #4a9eff;
}

.terminal-input-line {
    display: flex;
    align-items: flex-start;
}

.terminal-cursor {
    display: inline-block;
    width: 8px;
    height: 16px;
    background: #00ff00;
    animation: blink 1s infinite;
    margin-left: 2px;
}

@keyframes blink {
    0%, 49% { opacity: 1; }
    50%, 100% { opacity: 0; }
}

#hiddenInput {
    position: absolute;
    left: -9999px;
    opacity: 0;
}

.error {
    color: #ff6666;
}

.success {
    color: #66ff66;
}

.warning {
    color: #ffff66;
}

.info {
    color: #6666ff;
```

```
}

.dim {
    color: #888;
}

.highlight {
    color: #fff;
    background: #333;
    padding: 0 2px;
}

.header-ascii {
    color: #4a9eff;
    margin-bottom: 10px;
    font-size: 12px;
    line-height: 1.2;
}

pre {
    margin: 0;
    font-family: inherit;
}

.command {
    color: #ffaa00;
}

.ref {
    color: #ff66ff;
}

.resolved {
    color: #66ffff;
}

.bank-header {
    color: #ffff66;
    font-weight: bold;
}

.register-header {
    color: #ff9966;
    margin-left: 20px;
}

.address-line {
    margin-left: 40px;
}

.address-id {
    color: #9999ff;
}
```

```
.value {
    color: #66ff66;
}

/* File browser styling */
.file-list {
    background: #111;
    border: 1px solid #333;
    padding: 10px;
    margin: 10px 0;
    max-height: 200px;
    overflow-y: auto;
}

.file-item {
    padding: 5px;
    cursor: pointer;
    transition: background 0.2s;
}

.file-item:hover {
    background: #222;
}

.file-item.selected {
    background: #333;
    color: #4a9eff;
}

/* Status bar */
.status-bar {
    background: #111;
    border-top: 1px solid #333;
    padding: 5px 20px;
    display: flex;
    justify-content: space-between;
    font-size: 12px;
    color: #888;
}

.status-item {
    display: flex;
    align-items: center;
    gap: 10px;
}

.status-dirty {
    color: #ffaa00;
}

.modal {
    display: none;
    position: fixed;
    top: 0;
```

```
        left: 0;
        width: 100%;
        height: 100%;
        background: rgba(0, 0, 0, 0.8);
        z-index: 1000;
        align-items: center;
        justify-content: center;
    }

.modal.active {
    display: flex;
}

.modal-content {
    background: #1a1a1a;
    border: 2px solid #333;
    padding: 20px;
    max-width: 600px;
    width: 90%;
    max-height: 80vh;
    overflow-y: auto;
    color: #00ff00;
}

.modal-title {
    color: #4a9eff;
    margin-bottom: 15px;
    font-size: 16px;
}

#downloadLink {
    display: none;
}
</style>
</head>
<body>
    <div id="terminal"></div>
    <div class="status-bar">
        <div class="status-item">
            <span id="currentBank">No bank loaded</span>
            <span id="dirtyIndicator"></span>
        </div>
        <div class="status-item">
            <span id="configDisplay">prefix=x base=10</span>
        </div>
    </div>
    <input type="text" id="hiddenInput" autocomplete="off" autocorrect="off"
autocapitalize="off" spellcheck="false">
    <a id="downloadLink" download></a>

    <div class="modal" id="fileModal">
        <div class="modal-content">
            <div class="modal-title">Select File to Load</div>
            <div id="fileList" class="file-list"></div>
        </div>
    </div>
</body>
```

```
<div style="margin-top: 15px;">
    <button onclick="closeFileModal()">Cancel</button>
</div>
</div>

<script>
// CLIX JavaScript Implementation
class ClixConfig {
    constructor() {
        this.prefix = 'x';
        this.base = 10;
        this.width_bank = 5;
        this.width_reg = 2;
        this.width_addr = 4;
        this.allow_duplicate_values = false;
    }

    save() {
        localStorage.setItem('clix_config', JSON.stringify(this));
    }

    load() {
        const saved = localStorage.getItem('clix_config');
        if (saved) {
            Object.assign(this, JSON.parse(saved));
        }
    }

    toIniString() {
        return `prefix = ${this.prefix}
base = ${this.base}
width_bank = ${this.width_bank}
width_reg = ${this.width_reg}
width_addr = ${this.width_addr}
allow_duplicate_values = ${this.allow_duplicate_values}`;
    }

    fromIniString(ini) {
        const lines = ini.split('\n');
        for (const line of lines) {
            const match = line.match(/^(\w+)\s*=\s*(.+)$/);
            if (match) {
                const [_, key, value] = match;
                switch (key) {
                    case 'prefix':
                        this.prefix = value.trim();
                        break;
                    case 'base':
                        this.base = parseInt(value);
                        break;
                    case 'width_bank':
                        this.width_bank = parseInt(value);
                        break;
                }
            }
        }
    }
}
```

```
        case 'width_reg':
            this.width_reg = parseInt(value);
            break;
        case 'width_addr':
            this.width_addr = parseInt(value);
            break;
        case 'allow_duplicate_values':
            this.allow_duplicate_values = value.trim() ===
'true';
            break;
    }
}
}

class ClixBank {
constructor(id, title = '') {
    this.id = id;
    this.title = title;
    this.registers = {};
}

getRegister(regId) {
    if (!this.registers[regId]) {
        this.registers[regId] = {};
    }
    return this.registers[regId];
}

setValue(regId, addrId, value) {
    this.getRegister(regId)[addrId] = value;
}

getValue(regId, addrId) {
    return this.registers[regId]?.[addrId] || null;
}

deleteValue(regId, addrId) {
    if (this.registers[regId]) {
        delete this.registers[regId][addrId];
        if (Object.keys(this.registers[regId]).length === 0) {
            delete this.registers[regId];
        }
        return true;
    }
    return false;
}

toString(config) {
    let output = '';
    const bankStr = this.formatId(this.id, config.base,
config.width_bank);
    output += `${config.prefix}${bankStr} (${this.title}) {\n`;
}
```

```
const regIds = Object.keys(this.registers).map(Number).sort((a, b)
=> a - b);

for (const regId of regIds) {
    if (regIds.length > 1 || regId != 1) {
        const regStr = this.formatId(regId, config.base,
config.width_reg);
        output += `${regStr}\n`;
    }

    const addresses = this.registers[regId];
    const addrIds = Object.keys(addresses).map(Number).sort((a, b)
=> a - b);

    for (const addrId of addrIds) {
        const addrStr = this.formatId(addrId, config.base,
config.width_addr);
        output += `\t${addrStr}\t${addresses[addrId]}\n`;
    }
}

output += '}\n';
return output;
}

formatId(id, base, width) {
    let str = id.toString(base);
    return str.padStart(width, '0');
}

static fromString(text, config) {
    const lines = text.split('\n');
    let bank = null;
    let currentReg = 1;

    for (const line of lines) {
        const trimmed = line.trim();
        if (!trimmed) continue;

        // Header line
        const headerMatch = trimmed.match(new
RegExp(`^${config.prefix}([0-9a-zA-Z]+)\s*\`$`));
        if (headerMatch) {
            const id = parseInt(headerMatch[1], config.base);
            const title = headerMatch[2];
            bank = new ClixBank(id, title);
            continue;
        }

        // End of bank
        if (trimmed === '}') break;

        if (!bank) continue;
    }
}
```

```
// Register line
if (!line.startsWith('\t') && !line.startsWith(' ')) {
    const regMatch = trimmed.match(/^([0-9a-zA-Z]+)$/);
    if (regMatch) {
        currentReg = parseInt(regMatch[1], config.base);
    }
} else {
    // Address line
    const addrMatch = trimmed.match(/^([0-9a-zA-Z]+)\s+(.+)$/);
    if (addrMatch) {
        const addrId = parseInt(addrMatch[1], config.base);
        const value = addrMatch[2];
        bank.setValue(currentReg, addrId, value);
    }
}

return bank;
}

class ClixWorkspace {
constructor() {
    this.banks = {};
    this.currentBankId = null;
    this.dirty = false;
    this.config = new ClixConfig();
    this.resolver = new ClixResolver(this);
}

getBank(id) {
    return this.banks[id] || null;
}

createBank(id, title = '') {
    const bank = new ClixBank(id, title);
    this.banks[id] = bank;
    this.dirty = true;
    return bank;
}

loadBank(id) {
    const key = `clix_bank_${id}`;
    const saved = localStorage.getItem(key);
    if (saved) {
        try {
            const bank = ClixBank.fromString(saved, this.config);
            if (bank) {
                this.banks[id] = bank;
                return bank;
            }
        } catch (e) {

```

```
        console.error('Failed to load bank:', e);
    }
}
return null;
}

saveBank(id) {
    const bank = this.getBank(id);
    if (!bank) return false;

    const key = `clix_bank_${id}`;
    const content = bank.toString(this.config);
    localStorage.setItem(key, content);

    // Update saved banks list
    let savedBanks =
JSON.parse(localStorage.getItem('clix_saved_banks') || '[]');
    if (!savedBanks.includes(id)) {
        savedBanks.push(id);
        localStorage.setItem('clix_saved_banks',
JSON.stringify(savedBanks));
    }

    this.dirty = false;
    return true;
}

getValue(bankId, regId, addrId) {
    const bank = this.getBank(bankId);
    return bank ? bank.getValue(regId, addrId) : null;
}

setValue(bankId, regId, addrId, value) {
    if (!this.config.allow_duplicate_values &&
this.checkDuplicate(value)) {
        return { success: false, error: 'Duplicate value not allowed' };
    }

    let bank = this.getBank(bankId);
    if (!bank) {
        bank = this.createBank(bankId);
    }

    bank.setValue(regId, addrId, value);
    this.dirty = true;
    return { success: true };
}

deleteValue(bankId, regId, addrId) {
    const bank = this.getBank(bankId);
    if (bank) {
        const result = bank.deleteValue(regId, addrId);
        if (result) this.dirty = true;
    }
}
```

```
        return result;
    }
    return false;
}

checkDuplicate(value) {
    for (const bank of Object.values(this.banks)) {
        for (const reg of Object.values(bank.registers)) {
            for (const addrValue of Object.values(reg)) {
                if (addrValue === value) {
                    return true;
                }
            }
        }
    }
    return false;
}

getSavedBanks() {
    const savedBanks =
JSON.parse(localStorage.getItem('clix_saved_banks') || '[]');
    return savedBanks.map(id => {
        const bankStr =
id.toString(this.config.base).padStart(this.config.width_bank, '0');
        return {
            id,
            name: `${this.config.prefix}${bankStr}.txt`
        };
    });
}

exportBank(id) {
    const bank = this.getBank(id);
    if (!bank) return null;

    const content = bank.toString(this.config);
    const bankStr =
id.toString(this.config.base).padStart(this.config.width_bank, '0');
    const filename = `${this.config.prefix}${bankStr}.txt`;

    return { filename, content };
}

importBankFromText(text) {
    try {
        const bank = ClixBank.fromString(text, this.config);
        if (bank) {
            this.banks[bank.id] = bank;
            this.currentBankId = bank.id;
            this.dirty = true;
            return bank;
        }
    } catch (e) {
        console.error('Failed to import bank:', e);
    }
}
```

```
        }
        return null;
    }
}

class ClixResolver {
    constructor(workspace) {
        this.workspace = workspace;
        this.maxDepth = 16;
    }

    resolve(bankId, value, depth = 0, visited = new Set()) {
        if (depth >= this.maxDepth) {
            return '[Resolver Depth Exceeded]';
        }

        let output = '';
        let pos = 0;
        const config = this.workspace.config;

        while (pos < value.length) {
            let found = false;
            const remaining = value.substring(pos);

            // Pattern: x<bank>.<reg>.<addr>
            const fullRefPattern = new RegExp(`^${config.prefix}([0-9a-zA-Z]+)\\.([0-9a-zA-Z]+)\\\\.([0-9a-zA-Z]+)`);
            let match = remaining.match(fullRefPattern);

            if (match) {
                const bId = parseInt(match[1], config.base);
                const rId = parseInt(match[2], config.base);
                const aId = parseInt(match[3], config.base);

                const key = `${bId}.${rId}.${aId}`;
                if (visited.has(key)) {
                    output += '[Circular Ref]';
                } else {
                    const refValue = this.workspace.getValue(bId, rId, aId);
                    if (refValue !== null) {
                        const newVisited = new Set(visited);
                        newVisited.add(key);
                        output += this.resolve(bId, refValue, depth + 1, newVisited);
                    } else {
                        output += '[Missing Ref]';
                    }
                }
                pos += match[0].length;
                found = true;
            }
        }

        // Pattern: r<reg>.<addr> (same bank)
    }
}
```

```
if (!found) {
    match = remaining.match(/^r([0-9a-zA-Z]+)\.([0-9a-zA-Z]+)/);
    if (match) {
        const rId = parseInt(match[1], config.base);
        const aId = parseInt(match[2], config.base);

        const key = `${bankId}.${rId}.${aId}`;
        if (visited.has(key)) {
            output += '[Circular Ref]';
        } else {
            const refValue = this.workspace.getValue(bankId,
                rId, aId);
            if (refValue !== null) {
                const newVisited = new Set(visited);
                newVisited.add(key);
                output += this.resolve(bankId, refValue, depth
                    + 1, newVisited);
            } else {
                output += '[Missing Ref]';
            }
        }
    }
}

// Pattern: <bank>.<reg>.<addr> (numeric)
if (!found) {
    match = remaining.match(/^(\d+)\.(\d+)\.(\d+)/);
    if (match) {
        const bId = parseInt(match[1]);
        const rId = parseInt(match[2]);
        const aId = parseInt(match[3]);

        const key = `${bId}.${rId}.${aId}`;
        if (visited.has(key)) {
            output += '[Circular Ref]';
        } else {
            const refValue = this.workspace.getValue(bId, rId,
                aId);
            if (refValue !== null) {
                const newVisited = new Set(visited);
                newVisited.add(key);
                output += this.resolve(bId, refValue, depth +
                    1, newVisited);
            } else {
                output += '[Missing Ref]';
            }
        }
    }
}
pos += match[0].length;
found = true;
}
```

```
        if (!found) {
            output += value[pos];
            pos++;
        }
    }

    return output;
}

}

// Terminal REPL Implementation
class ClixREPL {
    constructor() {
        this.workspace = new ClixWorkspace();
        this.terminal = document.getElementById('terminal');
        this.hiddenInput = document.getElementById('hiddenInput');
        this.commandHistory = [];
        this.historyIndex = -1;
        this.currentInput = '';
        this.inputLine = null;
        this.cursorPosition = 0;

        this.init();
    }

    init() {
        this.workspace.config.load();
        this.printHeader();
        this.printLine('Type <span class="command">:help</span> for
commands.', false);
        this.printLine('');
        this.newLine();

        // Focus management
        document.addEventListener('click', () => {
            this.hiddenInput.focus();
        });

        // Keyboard input
        this.hiddenInput.addEventListener('input', (e) => {
            this.handleInput(e);
        });

        this.hiddenInput.addEventListener('keydown', (e) => {
            this.handleKeyDown(e);
        });

        // Keep focus
        this.hiddenInput.focus();

        this.updateStatusBar();
    }
}
```

```
    printHeader() {
        const header = `<pre class="header-ascii">
${this.terminal.asciiHeader}
</pre>`;
        this.printLine(header, false);
        this.printLine('<span class="info">CLIX HTML REPL - Safety-Critical Data System</span>', false);
        this.printLine('<span class="dim">Version 1.0.0 - Browser Edition</span>', false);
        this.printLine('');
    }

    printLine(text, escape = true) {
        const line = document.createElement('div');
        line.className = 'terminal-line';
        if (escape) {
            line.textContent = text;
        } else {
            line.innerHTML = text;
        }
        this.terminal.appendChild(line);
        this.scrollToBottom();
    }

    printError(text) {
        this.printLine(`Error: ${text}`, false);
    }

    printSuccess(text) {
        this.printLine(text, false);
    }

    newInputLine() {
        this.inputLine = document.createElement('div');
        this.inputLine.className = 'terminal-line terminal-input-line';
        this.updateInputLine();
        this.terminal.appendChild(this.inputLine);
        this.scrollToBottom();
    }

    updateInputLine() {
        const prompt = '<span class="terminal-prompt">&gt;&gt;</span> ';
        const beforeCursor =
this.escapeHtml(this.currentInput.substring(0, this.cursorPosition));
        const afterCursor =
this.escapeHtml(this.currentInput.substring(this.cursorPosition));
    }
}
```

```
        this.inputLine.innerHTML = prompt + beforeCursor + '<span  
class="terminal-cursor"></span>' + afterCursor;  
    }  
  
    escapeHtml(text) {  
        const div = document.createElement('div');  
        div.textContent = text;  
        return div.innerHTML;  
    }  
  
    scrollToBottom() {  
        this.terminal.scrollTop = this.terminal.scrollHeight;  
    }  
  
    handleInput(e) {  
        const newChar = e.data;  
        if (newChar) {  
            this.currentInput = this.currentInput.substring(0,  
this.cursorPosition) +  
                                newChar +  
  
            this.currentInput.substring(this.cursorPosition);  
            this.cursorPosition++;  
            this.updateInputLine();  
        }  
        this.hiddenInput.value = '^';  
    }  
  
    handleKeyDown(e) {  
        switch (e.key) {  
            case 'Enter':  
                e.preventDefault();  
                this.executeCommand();  
                break;  
  
            case 'Backspace':  
                e.preventDefault();  
                if (this.cursorPosition > 0) {  
                    this.currentInput = this.currentInput.substring(0,  
this.cursorPosition - 1) +  
  
                    this.currentInput.substring(this.cursorPosition);  
                    this.cursorPosition--;  
                    this.updateInputLine();  
                }  
                break;  
  
            case 'Delete':  
                e.preventDefault();  
                if (this.cursorPosition < this.currentInput.length) {  
                    this.currentInput = this.currentInput.substring(0,  
this.cursorPosition) +
```

```
this.currentInput.substring(this.cursorPosition + 1);
    this.updateInputLine();
}
break;

case 'ArrowLeft':
e.preventDefault();
if (this.cursorPosition > 0) {
    this.cursorPosition--;
    this.updateInputLine();
}
break;

case 'ArrowRight':
e.preventDefault();
if (this.cursorPosition < this.currentInput.length) {
    this.cursorPosition++;
    this.updateInputLine();
}
break;

case 'ArrowUp':
e.preventDefault();
if (this.historyIndex < this.commandHistory.length - 1) {
    this.historyIndex++;
    this.currentInput =
this.commandHistory[this.commandHistory.length - 1 - this.historyIndex];
    this.cursorPosition = this.currentInput.length;
    this.updateInputLine();
}
break;

case 'ArrowDown':
e.preventDefault();
if (this.historyIndex > 0) {
    this.historyIndex--;
    this.currentInput =
this.commandHistory[this.commandHistory.length - 1 - this.historyIndex];
    this.cursorPosition = this.currentInput.length;
    this.updateInputLine();
} else if (this.historyIndex === 0) {
    this.historyIndex = -1;
    this.currentInput = '';
    this.cursorPosition = 0;
    this.updateInputLine();
}
break;

case 'Home':
e.preventDefault();
this.cursorPosition = 0;
this.updateInputLine();
break;
```

```
        case 'End':
            e.preventDefault();
            this.cursorPosition = this.currentInput.length;
            this.updateInputLine();
            break;

        case 'Tab':
            e.preventDefault();
            this.handleTabComplete();
            break;
    }
}

handleTabComplete() {
    const commands = [':help', ':open', ':ls', ':show', ':ins',
':insr', ':del', ':delr',
                    ':w', ':run', ':resolve', ':plugins',
':plugin_run', ':toggle_dups',
                    ':q', ':export', ':import', ':clear', ':config'];

    if (this.currentInput.startsWith(':')) {
        const matches = commands.filter(cmd =>
cmd.startsWith(this.currentInput));
        if (matches.length === 1) {
            this.currentInput = matches[0] + ' ';
            this.cursorPosition = this.currentInput.length;
            this.updateInputLine();
        } else if (matches.length > 1) {
            this.printLine('');
            this.printLine('Available commands: ' + matches.join(','));
        }
        this.newLine();
        this.currentInput = this.currentInput;
        this.cursorPosition = this.currentInput.length;
        this.updateInputLine();
    }
}

executeCommand() {
    const command = this.currentInput.trim();
    if (!command) {
        this.printLine('>> ');
        this.newLine();
        return;
    }

    // Echo the command
    this.printLine(`>> ${command}`);

    // Add to history
    if (command !== this.commandHistory[this.commandHistory.length -
1]) {
        this.commandHistory.push(command);
    }
}
```

```
        if (this.commandHistory.length > 100) {
            this.commandHistory.shift();
        }
    }

    // Reset for new input
    this.currentInput = '';
    this.cursorPosition = 0;
    this.historyIndex = -1;

    // Parse and execute
    this.parseAndExecute(command);

    // New input line
    this.newInputLine();
}

parseAndExecute(command) {
    const parts = command.split(/\s+/);
    const cmd = parts[0];

    switch (cmd) {
        case ':help':
            this.showHelp();
            break;

        case ':open':
            if (parts.length >= 2) {
                this.openBank(parts[1]);
            } else {
                this.printError('Usage: :open <bank_id>');
            }
            break;

        case ':ls':
            this.listBanks();
            break;

        case ':show':
            this.showCurrentBank();
            break;

        case ':ins':
            if (parts.length >= 3) {
                const addr = parts[1];
                const value = parts.slice(2).join(' ');
                this.insertValue(1, addr, value);
            } else {
                this.printError('Usage: :ins <addr> <value>');
            }
            break;

        case ':insr':
            if (parts.length >= 4) {
```

```
        const reg = parts[1];
        const addr = parts[2];
        const value = parts.slice(3).join(' ');
        this.insertValue(reg, addr, value);
    } else {
        this.printError('Usage: :insr <reg> <addr> <value>');
    }
    break;

case ':del':
    if (parts.length >= 2) {
        this.deleteValue(1, parts[1]);
    } else {
        this.printError('Usage: :del <addr>');
    }
    break;

case ':delr':
    if (parts.length >= 3) {
        this.deleteValue(parts[1], parts[2]);
    } else {
        this.printError('Usage: :delr <reg> <addr>');
    }
    break;

case ':w':
    this.saveCurrentBank();
    break;

case ':run':
    if (parts.length >= 2) {
        this.runScript(parts[1]);
    } else {
        this.printError('Usage: :run <script.txt>');
    }
    break;

case ':resolve':
    this.showResolved();
    break;

case ':toggle_dups':
    this.toggleDuplicates();
    break;

case ':q':
    this.quit();
    break;

case ':clear':
    this.clearTerminal();
    break;

case ':export':
```

```

        this.exportCurrentBank();
        break;

        case ':import':
            this.importBank();
            break;

        case ':config':
            this.showConfig();
            break;

        case ':plugins':
            this.printLine('<span class="warning">Plugins not
available in browser version</span>', false);
            break;

        default:
            if (command.startsWith(':')) {
                this.printError(`Unknown command: ${cmd}. Type :help
for commands.`);
            } else {
                this.printError('Commands must start with ":"');
            }
        }
    }

    showHelp() {
        const help = `
CLIX HTML REPL Commands</span>
-----
-----
:help                                     Show this help
:open <ctx>                                Open/create
context (e.g., x00001 or 1)
:ls                                         List loaded contexts
:show                                       Print current buffer
:ins <addr> <value...>           Insert/replace into register 1
:insr <reg> <addr> <value...>      Insert/replace into a specific register
:del <addr>                                 Delete from
register 1
:delr <reg> <addr>                            Delete
from a specific register
:w                                         Write current buffer
to storage
:run <script.txt>                         Run commands
from a script (paste in prompt)
:resolve                                    Print resolved view of
current bank
:toggle_dups                               Toggle allowing
duplicate values
:export <file>                           Export current bank to
file
`;
    }
}

```

```
<span class="command">:import</span> Import bank from file
<span class="command">:config</span> Show configuration
<span class="command">:clear</span> Clear terminal
<span class="command">:q</span> Quit (clear session)
-----
-----
<span class="dim">Tab completion available for commands. Use arrow keys for history.</span>;

```

```
        this.printLine(help, false);
    }

    openBank(idStr) {
        let id;
        const config = this.workspace.config;

        // Remove .txt extension if present
        idStr = idStr.replace(/\.\txt$/, '');

        // Parse the ID
        if (idStr.startsWith(config.prefix)) {
            id = parseInt(idStr.substring(1), config.base);
        } else {
            id = parseInt(idStr, config.base);
        }

        if (isNaN(id)) {
            this.printError('Invalid bank ID format');
            return;
        }

        // Check if already loaded
        let bank = this.workspace.getBank(id);
        if (bank) {
            this.workspace.currentBankId = id;
            this.printSuccess(`<span class="success">Switched to loaded bank: ${config.prefix}${id.toString(config.base).padStart(config.width_bank, '0')}</span>`);
        } else {
            // Try to load from storage
            bank = this.workspace.loadBank(id);
            if (bank) {
                this.workspace.currentBankId = id;
                this.printSuccess(`<span class="success">Loaded bank from storage: ${config.prefix}${id.toString(config.base).padStart(config.width_bank, '0')}</span>`);
            } else {
                // Create new
                bank = this.workspace.createBank(id, `Bank ${id}`);
                this.workspace.currentBankId = id;
                this.printSuccess(`<span class="success">Created new bank: ${config.prefix}${id.toString(config.base).padStart(config.width_bank, '0')}</span>`);
            }
        }
    }
}
```

```
        }

        this.updateStatusBar();
    }

    listBanks() {
        const loaded = Object.keys(this.workspace.banks);
        const saved = this.workspace.getSavedBanks();

        if (loaded.length === 0 && saved.length === 0) {
            this.printLine('<span class="dim">(no banks loaded or saved)</span>', false);
            return;
        }

        if (loaded.length > 0) {
            this.printLine('<span class="info">Loaded banks:</span>', false);
            for (const id of loaded) {
                const bank = this.workspace.getBank(id);
                const config = this.workspace.config;
                const idStr =
id.toString(config.base).padStart(config.width_bank, '0');
                const current = id == this.workspace.currentBankId ? '<span class="warning">' + current + '</span>' : '';
                this.printLine(`<span class="bank-header">${config.prefix}${idStr}</span> (${bank.title})${current}`, false);
            }
        }

        if (saved.length > 0) {
            this.printLine('<span class="info">Saved banks:</span>', false);
            for (const item of saved) {
                if (!loaded.includes(item.id.toString())) {
                    this.printLine(`<span class="dim">${item.name}</span>`, false);
                }
            }
        }
    }

    showCurrentBank() {
        if (!this.workspace.currentBankId) {
            this.printError('No current bank. Use :open <bank_id>');
            return;
        }

        const bank = this.workspace.getBank(this.workspace.currentBankId);
        if (!bank) {
            this.printError('Current bank is invalid');
            return;
        }
    }
}
```

```
const output = bank.toString(this.workspace.config);
const lines = output.split('\n');

for (const line of lines) {
    if (line.includes('{') || line.includes('}')) {
        this.printLine(`<span class="bank-
header">${this.escapeHtml(line)}</span>`, false);
    } else if (line && !line.startsWith('\t')) {
        this.printLine(`<span class="register-
header">${this.escapeHtml(line)}</span>`, false);
    } else if (line) {
        const parts = line.trim().split(/\t+/);
        if (parts.length >= 2) {
            const addr = parts[0];
            const value = parts.slice(1).join('\t');
            this.printLine(`    <span class="address-
id">${this.escapeHtml(addr)}</span>    <span
class="value">${this.escapeHtml(value)}</span>`, false);
        } else {
            this.printLine(line);
        }
    }
}

insertValue(regStr, addrStr, value) {
    if (!this.workspace.currentBankId) {
        this.printError('No current bank. Use :open <bank_id>');
        return;
    }

    const config = this.workspace.config;
    const regId = parseInt(regStr, config.base);
    const addrId = parseInt(addrStr, config.base);

    if (isNaN(regId) || isNaN(addrId)) {
        this.printError('Invalid register or address ID');
        return;
    }

    const result =
this.workspace.setValue(this.workspace.currentBankId, regId, addrId, value);
    if (result.success) {
        this.printSuccess('<span class="success">Value
inserted</span>');
    } else {
        this.printError(result.error);
    }

    this.updateStatusBar();
}

deleteValue(regStr, addrStr) {
    if (!this.workspace.currentBankId) {
```

```
        this.printError('No current bank. Use :open <bank_id>');
        return;
    }

    const config = this.workspace.config;
    const regId = parseInt(regStr, config.base);
    const addrId = parseInt(addrStr, config.base);

    if (isNaN(regId) || isNaN(addrId)) {
        this.printError('Invalid register or address ID');
        return;
    }

    if (this.workspace.deleteValue(this.workspace.currentBankId,
regId, addrId)) {
        this.printSuccess('<span class="success">Value
deleted</span>');
    } else {
        this.printError('Value not found');
    }

    this.updateStatusBar();
}

saveCurrentBank() {
    if (!this.workspace.currentBankId) {
        this.printError('No current bank to save');
        return;
    }

    if (this.workspace.saveBank(this.workspace.currentBankId)) {
        this.printSuccess('<span class="success">Bank saved to
storage</span>');
    } else {
        this.printError('Failed to save bank');
    }

    this.updateStatusBar();
}

showResolved() {
    if (!this.workspace.currentBankId) {
        this.printError('No current bank. Use :open <bank_id>');
        return;
    }

    const bank = this.workspace.getBank(this.workspace.currentBankId);
    if (!bank) {
        this.printError('Current bank is invalid');
        return;
    }

    const config = this.workspace.config;
    const bankStr =
```

```
bank.id.toString(config.base).padStart(config.width_bank, '0');

        this.printLine(`<span class="info">Resolved view:</span>`, false);
        this.printLine(`<span class="bank-
header">${config.prefix}${bankStr} (${bank.title}) {</span>`, false);

    const regIds = Object.keys(bank.registers).map(Number).sort((a, b)
=> a - b);

        for (const regId of regIds) {
            if (regIds.length > 1 || regId != 1) {
                const regStr =
regId.toString(config.base).padStart(config.width_reg, '0');
                this.printLine(`<span class="register-header">${regStr}
</span>`, false);
            }
        }

        const addresses = bank.registers[regId];
        const addrIds = Object.keys(addresses).map(Number).sort((a, b)
=> a - b);

        for (const addrId of addrIds) {
            const addrStr =
addrId.toString(config.base).padStart(config.width_addr, '0');
            const value = addresses[addrId];
            const resolved = this.workspace.resolver.resolve(bank.id,
value);

            this.printLine(`      <span class="address-id">${addrStr}
</span>      <span class="resolved">${this.escapeHtml(resolved)}</span>`, false);
        }
    }

    this.printLine(`<span class="bank-header">}</span>`, false);
}

toggleDuplicates() {
    this.workspace.config.allow_duplicate_values =
!this.workspace.config.allow_duplicate_values;
    this.workspace.config.save();

    const status = this.workspace.config.allow_duplicate_values ?
'ALLOWED' : 'NOT ALLOWED';
    this.printSuccess(`<span class="success">Duplicate values are now:
${status}</span>`);

    this.updateStatusBar();
}

runScript(scriptName) {
    // In browser version, we'll prompt for script content
    const script = prompt(`Paste script content for
"${scriptName}":`);
    if (!script) {
```

```
        this.printError('Script cancelled');
        return;
    }

    const lines = script.split('\n');
    this.printLine(`<span class="info">Running script: ${scriptName}</span>`, false);

    for (const line of lines) {
        const trimmed = line.trim();
        if (!trimmed || trimmed.startsWith('#')) continue;

        this.printLine(`<span class="dim">&gt;&gt; ${trimmed}</span>`,
false);
        this.parseAndExecute(trimmed);
    }

    this.printLine('<span class="info">Script completed</span>',
false);
}

exportCurrentBank() {
    if (!this.workspace.currentBankId) {
        this.printError('No current bank to export');
        return;
    }

    const data =
this.workspace.exportBank(this.workspace.currentBankId);
    if (!data) {
        this.printError('Failed to export bank');
        return;
    }

    // Create download link
    const blob = new Blob([data.content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const link = document.getElementById('downloadLink');
    link.href = url;
    link.download = data.filename;
    link.click();

    setTimeout(() => URL.revokeObjectURL(url), 1000);

    this.printSuccess(`<span class="success">Bank exported as
${data.filename}</span>`);
}

importBank() {
    const input = document.createElement('input');
    input.type = 'file';
    input.accept = '.txt';

    input.onchange = (e) => {
```

```
const file = e.target.files[0];
if (!file) return;

const reader = new FileReader();
reader.onload = (event) => {
    const bank =
this.workspace.importBankFromText(event.target.result);
    if (bank) {
        this.printSuccess(`<span class="success">Imported
bank: ${bank.id} (${bank.title})</span>`);
        this.updateStatusBar();
    } else {
        this.printError('Failed to import bank - invalid
format');
    }
};

reader.readAsText(file);
};

input.click();
}

showConfig() {
    const config = this.workspace.config;
    this.printLine(`<span class="info">Current Configuration:</span>` ,
false);
    this.printLine(`  prefix = ${config.prefix}`);
    this.printLine(`  base = ${config.base}`);
    this.printLine(`  width_bank = ${config.width_bank}`);
    this.printLine(`  width_reg = ${config.width_reg}`);
    this.printLine(`  width_addr = ${config.width_addr}`);
    this.printLine(`  allow_duplicate_values =
${config.allow_duplicate_values}`);
}

clearTerminal() {
    this.terminal.innerHTML = '';
    this.printHeader();
    this.newLine();
}

quit() {
    if (this.workspace.dirty) {
        const confirm = prompt('Unsaved changes. Type "yes" to quit
anyway:');
        if (confirm !== 'yes') {
            this.printLine(`<span class="warning">Quit
cancelled</span>`, false);
            return;
        }
    }
}

this.printLine(`<span class="info">Clearing session...</span>` ,
false);
```

```
// Clear workspace
this.workspace = new ClixWorkspace();
this.workspace.config.load();
this.workspace.currentBankId = null;

    this.printLine('<span class="success">Session cleared. Type :help to start again.</span>', false);
    this.updateStatusBar();
}

updateStatusBar() {
    const bankDisplay = document.getElementById('currentBank');
    const dirtyDisplay = document.getElementById('dirtyIndicator');
    const configDisplay = document.getElementById('configDisplay');

    if (this.workspace.currentBankId) {
        const config = this.workspace.config;
        const idStr =
this.workspace.currentBankId.toString(config.base).padStart(config.width_bank,
'0');
        bankDisplay.textContent = `Bank: ${config.prefix}${idStr}`;
    } else {
        bankDisplay.textContent = 'No bank loaded';
    }

    if (this.workspace.dirty) {
        dirtyDisplay.textContent = '[unsaved]';
        dirtyDisplay.className = 'status-dirty';
    } else {
        dirtyDisplay.textContent = '';
    }

    const config = this.workspace.config;
    configDisplay.textContent = `prefix=${config.prefix}
base=${config.base} dupes=${config.allow_duplicate_values ? 'on' : 'off'}`;
}

closeFileModal() {
    document.getElementById('fileModal').classList.remove('active');
}
}

// Initialize REPL
const repl = new ClixREPL();
</script>
</body>
</html>
```

[19] GenesisOS/clix.js

- **Bytes:** 49835

- Type: text

```
// clix.js
export function initialize(gosApiProxy) {
    const container = gosApiProxy.window.getContainer();
    const doc = container.ownerDocument;

    // make the OS sandbox box behave like a normal full-height panel
    container.style.display = 'flex';
    container.style.flexDirection = 'column';
    container.style.width = '100%';
    container.style.height = '100%'; // parent window has a height, so we use it
    container.style.minHeight = '0'; // allow children to scroll
    container.style.overflow = 'hidden';
    // 1) inject scoped styles (based on original clix-repl.html)
:contentReference[oaicite:5]{index=5}
    const style = doc.createElement('style');
    style.textContent = `
        #clix-app {
            background: #0a0a0a;
            color: #00ff00;
            font-family: 'Consolas', 'Courier New', monospace;
            height: 100vh; /* ← fill the whole iframe viewport */
            max-height: 100vh; /* ← don't grow past it */
            box-sizing: border-box; /* ← padding won't push it past 100vh */
            display: flex;
            flex-direction: column;
            min-height: 0; /* ← lets #terminal scroll */
            overflow: hidden;
        }

        #clix-app #terminal {
            flex: 1 1 0; /* ← take all remaining space */
            padding: 20px;
            overflow: auto; /* ← vertical + horizontal */
            background: #000;
            position: relative;
            min-height: 0; /* ← actually enables scrolling in flex */
            min-width: 0;
        }

        #clix-app .terminal-line {
            white-space: pre; /* ← no wrapping, so horizontal scroll shows
*/
            line-height: 1.4;
        }

        #clix-app .terminal-prompt {
            color: #4a9eff;
        }
        #clix-app .terminal-input-line {
            display: flex;
        }
    `;
}
```

```
        align-items: flex-start;
    }
#clix-app .terminal-cursor {
    display: inline-block;
    width: 8px;
    height: 16px;
    background: #00ff00;
    animation: clix-blink 1s infinite;
    margin-left: 2px;
}
@keyframes clix-blink {
    0%, 49% { opacity: 1; }
    50%, 100% { opacity: 0; }
}
#clix-app #hiddenInput {
    position: absolute;
    left: -9999px;
    opacity: 0;
}
#clix-app .error { color: #ff6666; }
#clix-app .success { color: #66ff66; }
#clix-app .warning { color: #ffff66; }
#clix-app .info { color: #6666ff; }
#clix-app .dim { color: #888; }
#clix-app .highlight {
    color: #fff;
    background: #333;
    padding: 0 2px;
}
#clix-app .header-ascii {
    color: #4a9eff;
    margin-bottom: 10px;
    font-size: 12px;
    line-height: 1.2;
}
/* status bar stays at bottom */
#clix-app .status-bar {
    background: #111;
    border-top: 1px solid #333;
    padding: 5px 20px;
    display: flex;
    justify-content: space-between;
    font-size: 12px;
    color: #888;
    flex: 0 0 auto;
}
#clix-app .status-dirty { color: #ffaa00; }
#clix-app .status-item {
    display: flex;
    align-items: center;
    gap: 10px;
}
#clix-app .file-list {
```

```
background: #111;
border: 1px solid #333;
padding: 10px;
margin: 10px 0;
max-height: 200px;
overflow-y: auto;
}
#clix-app .file-item {
    padding: 5px;
    cursor: pointer;
    transition: background 0.2s;
}
#clix-app .file-item:hover {
    background: #222;
}
#clix-app .file-item.selected {
    background: #333;
    color: #4a9eff;
}
#clix-app .modal {
    display: none;
    position: fixed;
    inset: 0;
    background: rgba(0,0,0,0.8);
    z-index: 1000;
    align-items: center;
    justify-content: center;
}
#clix-app .modal.active { display: flex; }
#clix-app .modal-content {
    background: #1a1a1a;
    border: 2px solid #333;
    padding: 20px;
    max-width: 600px;
    width: 90%;
    max-height: 80vh;
    overflow-y: auto;
    color: #00ff00;
}
#clix-app .modal-title {
    color: #4a9eff;
    margin-bottom: 15px;
    font-size: 16px;
}
`;
doc.head.appendChild(style);

// 2) build the same HTML structure as clix-repl.html's <body>
:contentReference[oaicite:6]{index=6}
container.innerHTML =
<div id="clix-app">
    <div id="terminal"></div>
    <div class="status-bar">
        <div class="status-item">
```

```
        <span id="currentBank">No bank loaded</span>
        <span id="dirtyIndicator"></span>
    </div>
    <div class="status-item">
        <span id="configDisplay">prefix=x base=10</span>
    </div>
    </div>
    <input type="text" id="hiddenInput" autocomplete="off"
autocorrect="off" autocapitalize="off" spellcheck="false" />
    <a id="downloadLink" download style="display:none;"></a>
    <div class="modal" id="fileModal">
        <div class="modal-content">
            <div class="modal-title">Select File to Load</div>
            <div id="fileList" class="file-list"></div>
            <div style="margin-top: 15px;">
                <button id="clix-file-cancel">Cancel</button>
            </div>
        </div>
    </div>

    <div class="modal" id="promptModal">
        <div class="modal-content">
            <div class="modal-title" id="promptTitle">Input</div>
            <div style="margin-bottom: 10px;" id="promptMessage">Enter
value:</div>
            <input type="text" id="promptInput" style="width: 100%; background:#000; color:#0f0; border:1px solid #333; padding:4px;" />
            <div style="margin-top: 15px; display:flex; gap:10px; justify-content:flex-end;">
                <button id="promptCancel">Cancel</button>
                <button id="promptOk">OK</button>
            </div>
        </div>
    </div>

</div>
`;

const root = container.querySelector('#clix-app');

// ===== Original CLIX classes, slightly adapted to be container-scoped =====
class ClixConfig {
    constructor() {
        this.prefix = 'x';
        this.base = 10;
        this.width_bank = 5;
        this.width_reg = 2;
        this.width_addr = 4;
        this.allow_duplicate_values = false;
    }
    save() {
        localStorage.setItem('clix_config', JSON.stringify(this));
    }
}
```

```
load() {
    const saved = localStorage.getItem('clix_config');
    if (saved) Object.assign(this, JSON.parse(saved));
}
toIniString() {
    return `prefix = ${this.prefix}
base = ${this.base}
width_bank = ${this.width_bank}
width_reg = ${this.width_reg}
width_addr = ${this.width_addr}
allow_duplicate_values = ${this.allow_duplicate_values}`;
}
fromIniString(ini) {
    const lines = ini.split('\n');
    for (const line of lines) {
        const match = line.match(/^\s*(\w+)\s*=\s*(.+)$/);
        if (!match) continue;
        const [, key, value] = match;
        switch (key) {
            case 'prefix': this.prefix = value.trim(); break;
            case 'base': this.base = parseInt(value); break;
            case 'width_bank': this.width_bank = parseInt(value); break;
            case 'width_reg': this.width_reg = parseInt(value); break;
            case 'width_addr': this.width_addr = parseInt(value); break;
            case 'allow_duplicate_values': this.allow_duplicate_values =
value.trim() === 'true'; break;
        }
    }
}
}

class ClixBank {
constructor(id, title = '') {
    this.id = id;
    this.title = title;
    this.registers = {};
}
getRegister(regId) {
    if (!this.registers[regId]) this.registers[regId] = {};
    return this.registers[regId];
}
setValue(regId, addrId, value) {
    this.getRegister(regId)[addrId] = value;
}
getValue(regId, addrId) {
    return this.registers[regId]?.[addrId] || null;
}
deleteValue(regId, addrId) {
    if (!this.registers[regId]) return false;
    delete this.registers[regId][addrId];
    if (Object.keys(this.registers[regId]).length === 0) delete
this.registers[regId];
    return true;
}
}
```

```
formatId(id, base, width) {
    return id.toString(base).padStart(width, '0');
}
toString(config) {
    let output = '';
    const bankStr = this.formatId(this.id, config.base,
config.width_bank);
    output += `${config.prefix}${bankStr} (${this.title}) {\n`;
    const regIds = Object.keys(this.registers).map(Number).sort((a,b)=>a-
b);
    for (const regId of regIds) {
        if (regIds.length > 1 || regId !== 1) {
            const regStr = this.formatId(regId, config.base,
config.width_reg);
            output += `${regStr}\n`;
        }
        const addresses = this.registers[regId];
        const addrIds = Object.keys(addresses).map(Number).sort((a,b)=>a-
b);
        for (const addrId of addrIds) {
            const addrStr = this.formatId(addrId, config.base,
config.width_addr);
            output += `\t${addrStr}\t${addresses[addrId]}\n`;
        }
    }
    output += '}\n';
    return output;
}
static fromString(text, config) {
    const lines = text.split('\n');
    let bank = null;
    let currentReg = 1;
    for (const line of lines) {
        const trimmed = line.trim();
        if (!trimmed) continue;
        const headerMatch = trimmed.match(new RegExp(`^${config.prefix}
([0-9a-zA-Z]+)\s*\$`));
        if (headerMatch) {
            const id = parseInt(headerMatch[1], config.base);
            const title = headerMatch[2];
            bank = new ClixBank(id, title);
            continue;
        }
        if (trimmed === '}') break;
        if (!bank) continue;
        if (!line.startsWith('\t') && !line.startsWith(' ')) {
            const regMatch = trimmed.match(/^([0-9a-zA-Z]+)$/);
            if (regMatch) currentReg = parseInt(regMatch[1], config.base);
        } else {
            const addrMatch = trimmed.match(`^([0-9a-zA-Z]+)\s+(.+)$`);
            if (addrMatch) {
                const addrId = parseInt(addrMatch[1], config.base);
                const value = addrMatch[2];
                bank.setValue(currentReg, addrId, value);
            }
        }
    }
}
```

```
        }
    }
    return bank;
}

class ClixWorkspace {
    constructor() {
        this.banks = {};
        this.currentBankId = null;
        this.dirty = false;
        this.config = new ClixConfig();
        this.resolver = new ClixResolver(this);
    }
    getBank(id) { return this.banks[id] || null; }
    createBank(id, title='') {
        const bank = new ClixBank(id, title);
        this.banks[id] = bank;
        this.dirty = true;
        return bank;
    }
    loadBank(id) {
        const key = `clix_bank_${id}`;
        const saved = localStorage.getItem(key);
        if (!saved) return null;
        try {
            const bank = ClixBank.fromString(saved, this.config);
            if (bank) {
                this.banks[id] = bank;
                return bank;
            }
        } catch(e) { console.error(e); }
        return null;
    }
    saveBank(id) {
        const bank = this.getBank(id);
        if (!bank) return false;
        const key = `clix_bank_${id}`;
        localStorage.setItem(key, bank.toString(this.config));
        let savedBanks = JSON.parse(localStorage.getItem('clix_saved_banks'))
        || '[]';
        if (!savedBanks.includes(id)) {
            savedBanks.push(id);
            localStorage.setItem('clix_saved_banks',
JSON.stringify(savedBanks));
        }
        this.dirty = false;
        return true;
    }
    getValue(b, r, a) {
        const bank = this.getBank(b);
        return bank ? bank.getValue(r,a) : null;
    }
}
```

```
setValue(b, r, a, value) {
    if (!this.config.allow_duplicate_values && this.checkDuplicate(value))
{
    return { success:false, error:'Duplicate value not allowed' };
}
let bank = this.getBank(b);
if (!bank) bank = this.createBank(b);
bank.setValue(r,a,value);
this.dirty = true;
return { success:true };
}
deleteValue(b, r, a) {
    const bank = this.getBank(b);
    if (!bank) return false;
    const res = bank.deleteValue(r,a);
    if (res) this.dirty = true;
    return res;
}
checkDuplicate(value) {
    for (const b of Object.values(this.banks)) {
        for (const reg of Object.values(b.registers)) {
            for (const v of Object.values(reg)) {
                if (v === value) return true;
            }
        }
    }
    return false;
}
getSavedBanks() {
    const savedBanks = JSON.parse(localStorage.getItem('clix_saved_banks')
|| '[]');
    return savedBanks.map(id => {
        const bankStr =
id.toString(this.config.base).padStart(this.config.width_bank, '0');
        return { id, name: `${this.config.prefix}${bankStr}.txt` };
    });
}
exportBank(id) {
    const bank = this.getBank(id);
    if (!bank) return null;
    const content = bank.toString(this.config);
    const bankStr =
id.toString(this.config.base).padStart(this.config.width_bank, '0');
    const filename = `${this.config.prefix}${bankStr}.txt`;
    return { filename, content };
}
importBankFromText(text) {
    try {
        const bank = ClixBank.fromString(text, this.config);
        if (bank) {
            this.banks[bank.id] = bank;
            this.currentBankId = bank.id;
            this.dirty = true;
            return bank;
        }
    }
}
```

```
        }
    } catch(e){ console.error(e); }
    return null;
}

class ClixResolver {
    constructor(workspace) {
        this.workspace = workspace;
        this.maxDepth = 16;
    }
    resolve(bankId, value, depth=0, visited=new Set()) {
        if (depth >= this.maxDepth) return '[Resolver Depth Exceeded]';
        let output = '';
        let pos = 0;
        const config = this.workspace.config;
        while (pos < value.length) {
            let found = false;
            const remaining = value.substring(pos);

            // pattern: x<bank>.<reg>.<addr>
            const fullRefPattern = new RegExp(`^${config.prefix}([0-9a-zA-Z]+)\\.([0-9a-zA-Z]+)\\.( [0-9a-zA-Z]+)`);
            let match = remaining.match(fullRefPattern);
            if (match) {
                const bId = parseInt(match[1], config.base);
                const rId = parseInt(match[2], config.base);
                const aId = parseInt(match[3], config.base);
                const key = `${bId}.${rId}.${aId}`;
                if (visited.has(key)) {
                    output += '[Circular Ref]';
                } else {
                    const refValue = this.workspace.getValue(bId, rId, aId);
                    if (refValue !== null) {
                        const newVisited = new Set(visited);
                        newVisited.add(key);
                        output += this.resolve(bId, refValue, depth+1,
newVisited);
                    } else {
                        output += '[Missing Ref]';
                    }
                }
                pos += match[0].length;
                found = true;
            }

            // pattern: r<reg>.<addr> (same bank)
            if (!found) {
                match = remaining.match(/^r([0-9a-zA-Z]+)\\.([0-9a-zA-Z]+)/);
                if (match) {
                    const rId = parseInt(match[1], config.base);
                    const aId = parseInt(match[2], config.base);
                    const key = `${bankId}.${rId}.${aId}`;
                    if (visited.has(key)) {

```

```
        output += '[Circular Ref]';
    } else {
        const refValue = this.workspace.getValue(bankId, rId,
aId);
        if (refValue !== null) {
            const newVisited = new Set(visited);
            newVisited.add(key);
            output += this.resolve(bankId, refValue, depth+1,
newVisited);
        } else {
            output += '[Missing Ref]';
        }
    }
    pos += match[0].length;
    found = true;
}
}

// pattern: <bank>.<reg>.<addr> numeric
if (!found) {
    match = remaining.match(/^(\d+)\.(\d+)\.(\d+)/);
    if (match) {
        const bId = parseInt(match[1]);
        const rId = parseInt(match[2]);
        const aId = parseInt(match[3]);
        const key = `${bId}.${rId}.${aId}`;
        if (visited.has(key)) {
            output += '[Circular Ref]';
        } else {
            const refValue = this.workspace.getValue(bId, rId,
aId);
            if (refValue !== null) {
                const newVisited = new Set(visited);
                newVisited.add(key);
                output += this.resolve(bId, refValue, depth+1,
newVisited);
            } else {
                output += '[Missing Ref]';
            }
        }
    }
    pos += match[0].length;
    found = true;
}
}

if (!found) {
    output += value[pos];
    pos++;
}
}

return output;
}
}
```

```
class ClixREPL {
    constructor(root, doc) {
        this.root = root;
        this.doc = doc;
        this.workspace = new ClixWorkspace();
        this.terminal = root.querySelector('#terminal');
        this.hiddenInput = root.querySelector('#hiddenInput');
        this.commandHistory = [];
        this.historyIndex = -1;
        this.currentInput = '';
        this.inputLine = null;
        this.cursorPosition = 0;
        this.init();
    }

    init() {
        this.workspace.config.load();
        this.printHeader();
        this.printLine('Type <span class="command">:help</span> for
commands.', false);
        this.printLine('');
        this.newLine();

        // focus management, but scoped
        this.root.addEventListener('click', () => {
            this.hiddenInput.focus();
        });

        this.hiddenInput.addEventListener('input', e => this.handleInput(e));
        this.hiddenInput.addEventListener('keydown', e =>
this.handleKeyDown(e));
        this.hiddenInput.focus();

        // hook cancel in modal
        const cancelBtn = this.root.querySelector('#clix-file-cancel');
        if (cancelBtn) cancelBtn.addEventListener('click', () =>
this.closeFileModal());

        this.updateStatusBar();
    }

    printHeader() {
        const header = `<pre class="header-ascii">
${require('./header-ascii.txt')}
</pre>`;
    }
}
```

```
        this.printLine(header, false);
        this.printLine('<span class="info">CLIX HTML REPL - Safety-Critical
Data System</span>', false);
        this.printLine('<span class="dim">Version 1.0.0 - GOS Edition</span>',
false);
        this.printLine('');
    }

printLine(text, escape = true) {
    const line = this.doc.createElement('div');
    line.className = 'terminal-line';
    if (escape) line.textContent = text;
    else line.innerHTML = text;
    this.terminal.appendChild(line);
    this.scrollToBottom();
}

scrollToBottom() {
    this.terminal.scrollTop = this.terminal.scrollHeight;
}

newInputLine() {
    this.inputLine = this.doc.createElement('div');
    this.inputLine.className = 'terminal-line terminal-input-line';
    this.updateInputLine();
    this.terminal.appendChild(this.inputLine);
    this.scrollToBottom();
}

updateInputLine() {
    const prompt = '<span class="terminal-prompt">&gt;&gt;</span> ';
    const beforeCursor = this.escapeHtml(this.currentInput.substring(0,
this.cursorPosition));
    const afterCursor =
this.escapeHtml(this.currentInput.substring(this.cursorPosition));
    this.inputLine.innerHTML = prompt + beforeCursor + '<span
class="terminal-cursor"></span>' + afterCursor;
}

escapeHtml(text) {
    const div = this.doc.createElement('div');
    div.textContent = text;
    return div.innerHTML;
}

printError(msg) {
    const safe = this.escapeHtml(msg);
    const line = this.doc.createElement('div');
    line.className = 'terminal-line error';
    line.innerHTML = safe;
    this.terminal.appendChild(line);
    this.scrollToBottom();
}
```

```
handleInput(e) {
    const newChar = e.data;
    if (newChar) {
        this.currentInput = this.currentInput.slice(0,
this.cursorPosition) + newChar + this.currentInput.slice(this.cursorPosition);
        this.cursorPosition++;
        this.updateInputLine();
    }
    this.hiddenInput.value = '';
}

showPrompt(message, title = 'Input', defaultValue = '', cb) {
    const modal = this.root.querySelector('#promptModal');
    const titleEl = this.root.querySelector('#promptTitle');
    const msgEl = this.root.querySelector('#promptMessage');
    const inputEl = this.root.querySelector('#promptInput');
    const okBtn = this.root.querySelector('#promptOk');
    const cancelBtn = this.root.querySelector('#promptCancel');

    titleEl.textContent = title;
    msgEl.textContent = message;
    inputEl.value = defaultValue || '';
    modal.classList.add('active');
    inputEl.focus();

    const close = () => {
        modal.classList.remove('active');
        okBtn.onclick = null;
        cancelBtn.onclick = null;
    };

    okBtn.onclick = () => {
        const val = inputEl.value;
        close();
        if (cb) cb(val);
    };
    cancelBtn.onclick = () => {
        close();
        if (cb) cb(null);
    };
}

handleKeyDown(e) {
    switch (e.key) {
        case 'Enter':
            e.preventDefault();
            this.executeCommand();
            break;
        case 'Backspace':
            e.preventDefault();
            if (this.cursorPosition > 0) {
                this.currentInput = this.currentInput.slice(0,
this.cursorPosition - 1) + this.currentInput.slice(this.cursorPosition);
            }
    }
}
```

```
        this.cursorPosition--;
        this.updateInputLine();
    }
    break;
case 'Delete':
    e.preventDefault();
    if (this.cursorPosition < this.currentInput.length) {
        this.currentInput = this.currentInput.slice(0,
this.cursorPosition) + this.currentInput.slice(this.cursorPosition + 1);
        this.updateInputLine();
    }
    break;
case 'ArrowLeft':
    e.preventDefault();
    if (this.cursorPosition > 0) {
        this.cursorPosition--;
        this.updateInputLine();
    }
    break;
case 'ArrowRight':
    e.preventDefault();
    if (this.cursorPosition < this.currentInput.length) {
        this.cursorPosition++;
        this.updateInputLine();
    }
    break;
case 'ArrowUp':
    e.preventDefault();
    if (this.historyIndex < this.commandHistory.length - 1) {
        this.historyIndex++;
        this.currentInput =
this.commandHistory[this.commandHistory.length - 1 - this.historyIndex];
        this.cursorPosition = this.currentInput.length;
        this.updateInputLine();
    }
    break;
case 'ArrowDown':
    e.preventDefault();
    if (this.historyIndex > 0) {
        this.historyIndex--;
        this.currentInput =
this.commandHistory[this.commandHistory.length - 1 - this.historyIndex];
        this.cursorPosition = this.currentInput.length;
        this.updateInputLine();
    } else if (this.historyIndex === 0) {
        this.historyIndex = -1;
        this.currentInput = '';
        this.cursorPosition = 0;
        this.updateInputLine();
    }
    break;
case 'Home':
    e.preventDefault();
    this.cursorPosition = 0;
```

```
        this.updateInputLine();
        break;
    case 'End':
        e.preventDefault();
        this.cursorPosition = this.currentInput.length;
        this.updateInputLine();
        break;
    case 'Tab':
        e.preventDefault();
        this.handleTabComplete();
        break;
    }
}

handleTabComplete() {
    const commands = [':help', ':open', ':ls', ':show', ':ins', ':insr',
':del', ':delr',
':w', ':run', ':resolve', ':plugins', ':plugin_run',
':toggle_dups',
':q', ':export', ':import', ':clear', ':config'];
    if (this.currentInput.startsWith(':')) {
        const matches = commands.filter(c =>
c.startsWith(this.currentInput));
        if (matches.length === 1) {
            this.currentInput = matches[0] + ' ';
            this.cursorPosition = this.currentInput.length;
            this.updateInputLine();
        } else if (matches.length > 1) {
            this.printLine('');
            this.printLine('Available commands: ' + matches.join(', '));
            this.newInputLine();
            this.updateInputLine();
        }
    }
}

executeCommand() {
    const command = this.currentInput.trim();
    if (!command) {
        this.printLine('>> ');
        this.newInputLine();
        return;
    }

    this.printLine(`>> ${command}`);

    if (command !== this.commandHistory[this.commandHistory.length - 1]) {
        this.commandHistory.push(command);
        if (this.commandHistory.length > 100) this.commandHistory.shift();
    }

    this.currentInput = '';
    this.cursorPosition = 0;
    this.historyIndex = -1;
}
```

```
        this.parseAndExecute(command);
        this.newLine();
    }

parseAndExecute(command) {
    const parts = command.split(/\s+/);
    const cmd = parts[0];
    switch (cmd) {
        case ':help': this.showHelp(); break;
        case ':open':
            if (parts.length >= 2) this.openBank(parts[1]);
            else this.printError('Usage: :open <bank_id>');
            break;
        case ':ls': this.listBanks(); break;
        case ':show': this.showCurrentBank(); break;
        case ':ins':
            if (parts.length >= 3) {
                const addr = parts[1];
                const value = parts.slice(2).join(' ');
                this.insertValue(1, addr, value);
            } else this.printError('Usage: :ins <addr> <value>');
            break;
        case ':insr':
            if (parts.length >= 4) {
                const reg = parts[1];
                const addr = parts[2];
                const value = parts.slice(3).join(' ');
                this.insertValue(reg, addr, value);
            } else this.printError('Usage: :insr <reg> <addr> <value>');
            break;
        case ':del':
            if (parts.length >= 2) this.deleteValue(1, parts[1]);
            else this.printError('Usage: :del <addr>');
            break;
        case ':delr':
            if (parts.length >= 3) this.deleteValue(parts[1], parts[2]);
            else this.printError('Usage: :delr <reg> <addr>');
            break;
        case ':w': this.saveCurrentBank(); break;
        case ':run':
            if (parts.length >= 2) this.runScript(parts[1]);
            else this.printError('Usage: :run <script.txt>');
            break;
        case ':resolve': this.showResolved(); break;
        case ':toggle_dups': this.toggleDuplicates(); break;
        case ':q': this.quit(); break;
        case ':clear': this.clearTerminal(); break;
        case ':export': this.exportCurrentBank(); break;
        case ':import': this.importBank(); break;
        case ':config': this.showConfig(); break;
        case ':plugins':
            this.printLine('<span class="warning">Plugins not available in
browser/GOS version</span>', false);
    }
}
```

```

        break;
    default:
        if (command.startsWith(':')) this.printError(`Unknown command:
${cmd}. Type :help for commands.`);
        else this.printError('Commands must start with ":"');
    }
}

showHelp() {
    const help = `
CLIX HTML REPL Commands
-----
-----
:help Show this help
:open <ctx> Open/create
context (e.g. x00001 or 1)
:ls List loaded contexts
:show Print current buffer
:ins <addr> <value> Insert/replace into register 1
:insr <reg> <addr> <value> Insert/replace into a specific register
:del <addr> Delete from
register 1
:delr <reg> <addr> Delete
from a specific register
:w Write current buffer
to storage
:run <script.txt> Run commands
from a script (paste in prompt)
:resolve Print resolved view of
current bank
:toggle_dups Toggle allowing
duplicate values
:export Export current bank to
file
:import Import bank from file
:config Show configuration
:clear Clear terminal
:q Quit (clear session)
-----
-----
Tab completion available for commands. Use arrow keys for
history.`;
    this.printLine(help, false);
}

openBank(idStr) {
    const config = this.workspace.config;
    idStr = idStr.replace(/\.\txt$/, '');
    let id;
    if (idStr.startsWith(config.prefix)) id = parseInt(idStr.substring(1),
config.base);
    else id = parseInt(idStr, config.base);
}

```

```
if (isNaN(id)) {
    this.printError('Invalid bank ID format');
    return;
}
let bank = this.workspace.getBank(id);
if (bank) {
    this.workspace.currentBankId = id;
    this.printLine(`<span class="success">Switched to loaded bank:
${config.prefix}${id.toString(config.base).padStart(config.width_bank, '0')}
</span>`, false);
} else {
    bank = this.workspace.loadBank(id);
    if (bank) {
        this.workspace.currentBankId = id;
        this.printLine(`<span class="success">Loaded bank from
storage: ${config.prefix}${id.toString(config.base).padStart(config.width_bank,
'0')}</span>`, false);
    } else {
        bank = this.workspace.createBank(id, `Bank ${id}`);
        this.workspace.currentBankId = id;
        this.printLine(`<span class="success">Created new bank:
${config.prefix}${id.toString(config.base).padStart(config.width_bank, '0')}
</span>`, false);
    }
}
this.updateStatusBar();
}

listBanks() {
    const loaded = Object.keys(this.workspace.banks);
    const saved = this.workspace.getSavedBanks();
    if (loaded.length === 0 && saved.length === 0) {
        this.printLine(`<span class="dim">(no banks loaded or saved)
</span>`, false);
        return;
    }
    if (loaded.length > 0) {
        this.printLine(`<span class="info">Loaded banks:</span>`, false);
        for (const id of loaded) {
            const bank = this.workspace.getBank(id);
            const config = this.workspace.config;
            const idStr =
id.toString(config.base).padStart(config.width_bank, '0');
            const current = id === this.workspace.currentBankId ? `<span
class="warning">[current]</span>` : '';
            this.printLine(`<span class="bank-
header">${config.prefix}${idStr}</span> (${bank.title})${current}`, false);
        }
    }
    if (saved.length > 0) {
        this.printLine(`<span class="info">Saved banks:</span>`, false);
        for (const item of saved) {
            if (!loaded.includes(item.id.toString())) {
                this.printLine(`<span class="dim">${item.name}</span>`,

```

```
false);
        }
    }
}

showCurrentBank() {
    if (!this.workspace.currentBankId) {
        this.printError('No bank loaded');
        return;
    }
    const bank = this.workspace.getBank(this.workspace.currentBankId);
    const config = this.workspace.config;
    this.printLine(`<span class="bank-
header">${config.prefix}${this.workspace.currentBankId.toString(config.base).padStart(config.width_bank, '0')} ${{bank.title}} {</span>` , false);
    const regIds = Object.keys(bank.registers).map(Number).sort((a,b)=>a-
b);
    for (const regId of regIds) {
        if (regIds.length > 1 || regId !== 1) {
            this.printLine(`<span class="register-
header">${regId.toString(config.base).padStart(config.width_reg, '0')}</span>` ,
false);
        }
        const addresses = bank.registers[regId];
        const addrIds = Object.keys(addresses).map(Number).sort((a,b)=>a-
b);
        for (const addrId of addrIds) {
            const addrStr =
addrId.toString(config.base).padStart(config.width_addr, '0');
            this.printLine(`\t<span class="address-id">${addrStr}</span>\t<span class="value">${addresses[addrId]}</span>` , false);
        }
    }
    this.printLine('<span class="bank-header">}</span>' , false);
}

insertValue(reg, addr, value) {
    if (!this.workspace.currentBankId) {
        this.printError('No bank loaded. Use :open <id> first.');
        return;
    }
    const config = this.workspace.config;
    const regId = parseInt(reg, config.base);
    const addrId = parseInt(addr, config.base);
    if (isNaN(regId) || isNaN(addrId)) {
        this.printError('Invalid reg/addr format');
        return;
    }
    const res = this.workspace.setValue(this.workspace.currentBankId,
regId, addrId, value);
    if (!res.success) {
        this.printError(res.error);
    } else {
```

```
        this.printLine('<span class="success">Value inserted.</span>',  
false);  
        this.updateStatusBar();  
    }  
}  
  
deleteValue(reg, addr) {  
    if (!this.workspace.currentBankId) {  
        this.printError('No bank loaded.');//  
        return;  
    }  
    const config = this.workspace.config;  
    const regId = parseInt(reg, config.base);  
    const addrId = parseInt(addr, config.base);  
    if (isNaN(regId) || isNaN(addrId)) {  
        this.printError('Invalid reg/addr format');//  
        return;  
    }  
    const ok = this.workspace.deleteValue(this.workspace.currentBankId,  
regId, addrId);  
    if (ok) {  
        this.printLine('<span class="success">Value deleted.</span>',  
false);  
        this.updateStatusBar();  
    } else {  
        this.printError('No such value to delete.');//  
    }  
}  
  
saveCurrentBank() {  
    if (!this.workspace.currentBankId) {  
        this.printError('No bank loaded.');//  
        return;  
    }  
    const ok = this.workspace.saveBank(this.workspace.currentBankId);  
    if (ok) this.printLine('<span class="success">Bank saved.</span>',  
false);  
    else this.printError('Failed to save bank.');//  
    this.updateStatusBar();  
}  
  
showResolved() {  
    if (!this.workspace.currentBankId) {  
        this.printError('No bank loaded.');//  
        return;  
    }  
    const bank = this.workspace.getBank(this.workspace.currentBankId);  
    const config = this.workspace.config;  
    this.printLine('<span class="info">Resolved view:</span>', false);  
    const regIds = Object.keys(bank.registers).map(Number).sort((a,b)=>a-  
b);  
    for (const regId of regIds) {  
        const addrObj = bank.registers[regId];  
        const addrIds = Object.keys(addrObj).map(Number).sort((a,b)=>a-b);  
    }  
}
```

```
        for (const addrId of addrIds) {
            const rawVal = addrObj[addrId];
            const resolved =
this.workspace.resolver.resolve(this.workspace.currentBankId, rawVal);
            const addrStr =
addrId.toString(config.base).padStart(config.width_addr, '0');
            this.printLine(`<span class="address-id">${addrStr}
</span>\t<span class="value">${this.escapeHtml(resolved)}</span>`, false);
        }
    }

toggleDuplicates() {
    this.workspace.config.allow_duplicate_values =
!this.workspace.config.allow_duplicate_values;
    this.workspace.config.save();
    const status = this.workspace.config.allow_duplicate_values ?
'ALLOWED' : 'NOT ALLOWED';
    this.printLine(`<span class="success">Duplicate values are now:
${status}</span>`, false);
    this.updateStatusBar();
}

runScript(name) {
    this.showPrompt(
        `Paste script content for "${name}" (linebreaks allowed if you
paste):`,
        'Run Script',
        '',
        (script) => {
            if (!script) {
                this.printError('Script cancelled');
                return;
            }
            this.printLine(`<span class="info">Running script: ${name}
</span>`, false);
            const lines = script.split('\n');
            for (const line of lines) {
                const trimmed = line.trim();
                if (!trimmed || trimmed.startsWith('#')) continue;
                this.printLine(`<span class="dim">&gt;&gt; ${trimmed}
</span>`, false);
                this.parseAndExecute(trimmed);
            }
            this.printLine(`<span class="info">Script completed</span>`,
false);
        }
    );
}

exportCurrentBank() {
    if (!this.workspace.currentBankId) {
        this.printError('No current bank to export');
        return;
    }
}
```

```
        }
        const data = this.workspace.exportBank(this.workspace.currentBankId);
        if (!data) {
            this.printError('Failed to export bank');
            return;
        }
        const blob = new Blob([data.content], { type: 'text/plain' });
        const url = URL.createObjectURL(blob);
        const link = this.root.querySelector('#downloadLink');
        link.href = url;
        link.download = data.filename;
        link.click();
        setTimeout(() => URL.revokeObjectURL(url), 1000);
        this.printLine(`<span class="success">Bank exported as
${data.filename}</span>`, false);
    }

    importBank() {
        const input = this.doc.createElement('input');
        input.type = 'file';
        input.accept = '.txt';
        input.onchange = (e) => {
            const file = e.target.files[0];
            if (!file) return;
            const reader = new FileReader();
            reader.onload = (evt) => {
                const bank =
this.workspace.importBankFromText(evt.target.result);
                if (bank) {
                    this.printLine(`<span class="success">Imported bank:
${bank.id} (${bank.title})</span>`, false);
                    this.updateStatusBar();
                } else {
                    this.printError('Failed to import bank - invalid format');
                }
            };
            reader.readAsText(file);
        };
        input.click();
    }

    showConfig() {
        const config = this.workspace.config;
        this.printLine(`<span class="info">Current Configuration:</span>`,
false);
        this.printLine(` prefix = ${config.prefix}`);
        this.printLine(` base = ${config.base}`);
        this.printLine(` width_bank = ${config.width_bank}`);
        this.printLine(` width_reg = ${config.width_reg}`);
        this.printLine(` width_addr = ${config.width_addr}`);
        this.printLine(` allow_duplicate_values =
${config.allow_duplicate_values}`);
    }
}
```

```
clearTerminal() {
    this.terminal.innerHTML = '';
    this.printHeader();
    this.newLine();
}

quit() {
    if (this.workspace.dirty) {
        this.showPrompt(
            'Unsaved changes. Type "yes" to quit anyway:',
            'Confirm Quit',
            '',
            (val) => {
                if (val !== 'yes') {
                    this.printLine('<span class="warning">Quit
cancelled</span>', false);
                    return;
                }
                this._doQuit();
            }
        );
    } else {
        this._doQuit();
    }
}

_doQuit() {
    this.printLine('<span class="info">Clearing session...</span>',
false);
    this.workspace = new ClixWorkspace();
    this.workspace.config.load();
    this.workspace.currentBankId = null;
    this.printLine('<span class="success">Session cleared. Type :help to
start again.</span>', false);
    this.updateStatusBar();
}

updateStatusBar() {
    const bankDisplay = this.root.querySelector('#currentBank');
    const dirtyDisplay = this.root.querySelector('#dirtyIndicator');
    const configDisplay = this.root.querySelector('#configDisplay');
    if (this.workspace.currentBankId) {
        const config = this.workspace.config;
        const idStr =
this.workspace.currentBankId.toString(config.base).padStart(config.width_bank,
'0');
        bankDisplay.textContent = `Bank: ${config.prefix}${idStr}`;
    } else {
        bankDisplay.textContent = 'No bank loaded';
    }
    if (this.workspace.dirty) {
        dirtyDisplay.textContent = '[unsaved]';
        dirtyDisplay.className = 'status-dirty';
    }
}
```

```

    } else {
        dirtyDisplay.textContent = '';
        dirtyDisplay.className = '';
    }
    const config = this.workspace.config;
    configDisplay.textContent = `prefix=${config.prefix}
base=${config.base} dupes=${config.allow_duplicate_values ? 'on' : 'off'}`;
}

closeFileModal() {
    const modal = this.root.querySelector('#fileModal');
    if (modal) modal.classList.remove('active');
}
}

// 3) boot CLIX
new ClixREPL(container.querySelector('#clix-app'), doc);
}

```

## [20] GenesisOS/clix.json

- **Bytes:** 425
- **Type:** text

```
{
  "id": "clix",
  "title": "CLIX REPL System",
  "description": "Safety-critical data bank REPL (CLIX) with localStorage banks, resolver, export/import – packaged for GOS.",
  "version": "1.0.0",
  "author": "Dominic Alexander Cooper",
  "icon": "■",
  "category": "Development",
  "entry": "clix.js",
  "permissions": [],
  "window": {
    "width": 1100,
    "height": 650,
    "resizable": true
  }
}
```

## [21] GenesisOS/config.php

- **Bytes:** 957
- **Type:** text

```

<?php
/**
 * Text Reversion
 * Configuration File
 */

return [
    'system' => [
        'name' => 'Genesis OS',
        'version' => '1.0.0',
        'build' => '20250704',
        'debug' => true
    ],
    'security' => [
        //  NEW: Add a flag to control developer registration.
        'allow_developer_registration' => true,
        'sandbox_enabled' => true,
        'sandbox_memory_limit' => '64M',
        'sandbox_time_limit' => 5 // seconds
    ],
    'filesystem' => [
        'use_local' => true,
        'root_dir' => __DIR__ . '/gos_files',
        'user_dir' => __DIR__ . '/gos_files/users',
        'apps_dir' => __DIR__ . '/gos_files/apps',
        'system_dir' => __DIR__ . '/gos_files/system',
        'temp_dir' => __DIR__ . '/gos_files/temp'
    ],
    'ui' => [
        'theme' => 'vintage',
        'animations' => true,
        'fontSize' => 'medium',
        'language' => 'en'
    ]
];

```

## [22] GenesisOS/databank-manager.js

- **Bytes:** 54480
- **Type:** text

```

export function initialize(GOS) {
    const doc = GOS.window.getContainer().ownerDocument;
    const body = doc.body;

    // 1. Inject the application's CSS styles
    const styleEl = doc.createElement('style');
    styleEl.textContent =
        /* algorithm-themes.css */

```

```
/* Theme Selection Variables - Default Theme */
:root {
    /* Background Colors */
    --bg-primary: #f0f0f0;
    --bg-container: #fff;
    --bg-header: #fff;
    --bg-section-title: #f5f5f5;
    --bg-tab: #f5f5f5;
    --bg-tab-active: #fff;
    --bg-toolbar: #f9f9f9;
    --bg-register-header: #f5f5f5;
    --bg-table-header: #f9f9f9;
    --bg-multi-line: #f9f9f9;
    --bg-toast: #333;
    --bg-highlight: #fff3cd;
    --bg-editor: #fff;
    /* Text Colors */
    --text-primary: #333;
    --text-secondary: #666;
    --text-contrast: #fff;
    --text-reference: #0066cc;
    --text-resolved: #0066cc;
    /* Border Colors */
    --border-light: #ddd;
    --border-medium: #ccc;
    --border-focus: #4d90fe;
    --border-multi-line: #ddd;
    /* Button Colors */
    --btn-bg: #f0f0f0;
    --btn-hover-bg: #e0e0e0;
    --btn-border: #ccc;
    --btn-text: #333;
    /* Font Families */
    --font-primary: 'Arial', sans-serif;
    --font-mono: monospace;

    /* Shadows */
    --shadow-sm: 0 2px 5px rgba(0,0,0,0.1);
    --shadow-md: 0 3px 6px rgba(0,0,0,0.2);
    --shadow-lg: 0 3px 10px rgba(0,0,0,0.2);
    /* Animation */
    --transition-speed: 0.3s;
}
/* C++ Developer Theme */
.theme-cpp {
    /* Background Colors */
    --bg-primary: #1e1e1e;
    --bg-container: #252526;
    --bg-header: #323233;
    --bg-section-title: #37373d;
    --bg-tab: #2d2d2d;
    --bg-tab-active: #1e1e1e;
    --bg-toolbar: #333333;
    --bg-register-header: #3c3c3c;
```

```
--bg-table-header: #2d2d2d;
--bg-multi-line: #1e1e1e;
--bg-toast: #0e639c;
--bg-highlight: #264f78;
--bg-editor: #1e1e1e;
/* Text Colors */
--text-primary: #d4d4d4;
--text-secondary: #bbbbbb;
--text-contrast: #ffffff;
--text-reference: #569cd6;
--text-resolved: #4ec9b0;
/* Border Colors */
--border-light: #3c3c3c;
--border-medium: #555555;
--border-focus: #007acc;
--border-multi-line: #3c3c3c;
/* Button Colors */
--btn-bg: #3c3c3c;
--btn-hover-bg: #505050;
--btn-border: #6a6a6a;
--btn-text: #cccccc;
/* Font Families */
--font-primary: 'Consolas', 'Courier New', monospace;
--font-mono: 'Consolas', 'Courier New', monospace;

/* Shadows */
--shadow-sm: 0 2px 5px rgba(0,0,0,0.3);
--shadow-md: 0 3px 6px rgba(0,0,0,0.4);
--shadow-lg: 0 3px 10px rgba(0,0,0,0.5);
}

/* Dark Theme */
.theme-dark {
    /* Background Colors */
    --bg-primary: #121212;
    --bg-container: #1e1e1e;
    --bg-header: #252525;
    --bg-section-title: #2a2a2a;
    --bg-tab: #252525;
    --bg-tab-active: #333333;
    --bg-toolbar: #2a2a2a;
    --bg-register-header: #303030;
    --bg-table-header: #252525;
    --bg-multi-line: #1a1a1a;
    --bg-toast: #424242;
    --bg-highlight: #2d4f50;
    --bg-editor: #1e1e1e;
    /* Text Colors */
    --text-primary: #e0e0e0;
    --text-secondary: #b0b0b0;
    --text-contrast: #ffffff;
    --text-reference: #90caf9;
    --text-resolved: #80cbc4;
    /* Border Colors */
    --border-light: #424242;
```

```
--border-medium: #555555;
--border-focus: #64b5f6;
--border-multi-line: #424242;
/* Button Colors */
--btn-bg: #424242;
--btn-hover-bg: #505050;
--btn-border: #616161;
--btn-text: #e0e0e0;
}
/* Light Theme */
.theme-light {
    /* Background Colors */
    --bg-primary: #ffffff;
    --bg-container: #ffffff;
    --bg-header: #f8f9fa;
    --bg-section-title: #f1f3f4;
    --bg-tab: #f1f3f4;
    --bg-tab-active: #ffffff;
    --bg-toolbar: #f8f9fa;
    --bg-register-header: #f1f3f4;
    --bg-table-header: #f8f9fa;
    --bg-multi-line: #f8f9fa;
    --bg-toast: #5f6368;
    --bg-highlight: #e8f0fe;
    --bg-editor: #ffffff;
    /* Text Colors */
    --text-primary: #202124;
    --text-secondary: #5f6368;
    --text-contrast: #ffffff;
    --text-reference: #1a73e8;
    --text-resolved: #188038;
    /* Border Colors */
    --border-light: #dadce0;
    --border-medium: #bdc1c6;
    --border-focus: #1a73e8;
    --border-multi-line: #dadce0;
    /* Button Colors */
    --btn-bg: #f1f3f4;
    --btn-hover-bg: #e8eaed;
    --btn-border: #dadce0;
    --btn-text: #202124;
}
/* Theme transition effect */
body, body * {
    transition: background-color var(--transition-speed),
                color var(--transition-speed),
                border-color var(--transition-speed),
                box-shadow var(--transition-speed);
}
/* Global styles */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
```

```
        font-family: var(--font-primary);
    }
body {
    background-color: var(--bg-primary);
    padding: 10px;
    overflow-y: auto;
    min-height: 100vh;
    color: var(--text-primary);
}
.app-container {
    max-width: 100%;
    margin: 0 auto;
    background-color: var(--bg-container);
    border-radius: 5px;
    box-shadow: var(--shadow-sm);
    display: flex;
    flex-direction: column;
    min-height: calc(100vh - 20px);
    overflow: hidden;
}
/* Theme Selector */
.theme-selector {
    position: absolute;
    top: 10px;
    right: 15px;
    display: flex;
    align-items: center;
    z-index: 10;
}
.theme-selector label {
    margin-right: 10px;
    color: var(--text-primary);
    font-size: 0.9rem;
}
.theme-selector select {
    padding: 5px;
    border: 1px solid var(--border-medium);
    border-radius: 3px;
    background-color: var(--bg-container);
    color: var(--text-primary);
}
/* Main content area - allow scrolling */
.main-content {
    flex: 1;
    overflow-y: auto;
    -webkit-overflow-scrolling: touch; /* Smooth scrolling on iOS */
}
/* Header styles */
header {
    text-align: center;
    padding: 15px;
    border-bottom: 1px solid var(--border-light);
    background-color: var(--bg-header);
}
```

```
header h1 {
    font-size: 1.5rem;
    font-weight: bold;
    margin-bottom: 5px;
    color: var(--text-primary);
}
header p {
    color: var(--text-secondary);
    font-size: 0.9rem;
}
/* Section styles */
section {
    padding: 10px;
    border-bottom: 1px solid var(--border-light);
}
section h2 {
    font-size: 1rem;
    margin-bottom: 10px;
    padding: 5px;
    background-color: var(--bg-section-title);
    border: 1px solid var(--border-light);
    border-radius: 3px;
    color: var(--text-primary);
}
/* Configuration section */
.config-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
    gap: 10px;
    margin-bottom: 10px;
}
.config-item {
    display: flex;
    align-items: center;
}
.config-item label {
    flex: 1;
    font-size: 0.9rem;
    color: var(--text-primary);
}
.config-item input,
.config-item select {
    width: 100px;
    padding: 5px;
    border: 1px solid var(--border-medium);
    border-radius: 3px;
    background-color: var(--bg-container);
    color: var(--text-primary);
}
.file-actions {
    display: flex;
    align-items: center;
    flex-wrap: wrap;
    gap: 10px;
}
```

```
        padding: 10px 0;
    }
    .file-actions label {
        margin-right: 10px;
        color: var(--text-primary);
    }
    /* Button styles */
    button {
        padding: 5px 10px;
        background-color: var(--btn-bg);
        border: 1px solid var(--btn-border);
        border-radius: 3px;
        cursor: pointer;
        font-size: 0.8rem;
        white-space: nowrap;
        color: var(--btn-text);
    }
    button:hover {
        background-color: var(--btn-hover-bg);
    }
    button:focus {
        outline: 2px solid var(--border-focus);
    }
    /* Primary button style */
    .primary-button {
        font-weight: bold;
    }
    /* Banks section */
    .banks-section {
        flex: 1;
        display: flex;
        flex-direction: column;
        overflow: hidden;
    }
    .tab-container {
        display: flex;
        border-bottom: 1px solid var(--border-light);
        overflow-x: auto;
        -webkit-overflow-scrolling: touch;
        scrollbar-width: thin;
    }
    .tab {
        padding: 8px 15px;
        background-color: var(--bg-tab);
        border: 1px solid var(--border-medium);
        border-bottom: none;
        margin-right: 1px;
        cursor: pointer;
        border-radius: 3px 3px 0 0;
        white-space: nowrap;
        color: var(--text-primary);
        position: relative;
        padding-right: 25px;
    }
```

```
.tab:focus {
    outline: 2px solid var(--border-focus);
}
.tab.active {
    background-color: var(--bg-tab-active);
    border-bottom: 1px solid var(--bg-tab-active);
    margin-bottom: -1px;
}
.tab-title {
    font-size: 0.9rem;
}
.tab-close {
    position: absolute;
    right: 5px;
    top: 50%;
    transform: translateY(-50%);
    background: none;
    border: none;
    font-size: 14px;
    cursor: pointer;
    width: 16px;
    height: 16px;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 50%;
    opacity: 0.5;
    transition: all 0.2s;
}
.tab:hover .tab-close,
.tab.active .tab-close {
    opacity: 0.8;
}
.tab-close:hover {
    background-color: rgba(255,0,0,0.2);
    opacity: 1;
}
.tab-contents {
    flex: 1;
    overflow: auto;
}
.tab-content {
    padding: 10px;
    display: none;
    height: 100%;
}
.tab-content.active {
    display: flex;
    flex-direction: column;
}
/* Bank display */
.bank-toolbar {
    display: flex;
    justify-content: space-between;
```

```
margin-bottom: 10px;
padding: 5px;
background-color: var(--bg-toolbar);
border: 1px solid var(--border-light);
border-radius: 3px;
color: var(--text-primary);
}
.toolbar-title {
  display: flex;
  align-items: center;
  gap: 10px;
}
.toolbar-actions {
  display: flex;
  gap: 5px;
}
.toolbar-info {
  display: flex;
  flex-direction: column;
}
.toolbar-controls {
  display: flex;
  align-items: center;
  gap: 10px;
}
.bank-title {
  margin: 0;
}
.bank-stats {
  font-size: 0.8em;
  opacity: 0.7;
}
.edit-hint {
  font-size: 0.8em;
  opacity: 0.7;
}
registers-container {
  overflow-y: auto;
  flex: 1;
}
register-container {
  margin-bottom: 20px;
}
.register-container.collapsed .table-container {
  display: none;
}
.register-header {
  background-color: var(--bg-register-header);
  padding: 5px 10px;
  border: 1px solid var(--border-light);
  border-radius: 3px 3px 0 0;
  font-weight: bold;
  color: var(--text-primary);
  cursor: pointer;
```

```
        display: flex;
        justify-content: space-between;
        align-items: center;
    }
.register-title {
    font-weight: bold;
    display: flex;
    align-items: center;
    gap: 5px;
}
.collapse-icon {
    font-size: 0.8em;
    font-family: monospace;
}
.register-count {
    font-size: 0.8em;
    opacity: 0.7;
}
.addresses-table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 5px;
}
.addresses-table th,
.addresses-table td {
    text-align: left;
    padding: 5px 10px;
    border: 1px solid var(--border-light);
    color: var(--text-primary);
}
.addresses-table th {
    background-color: var(--bg-table-header);
    font-weight: normal;
}
.sortable {
    cursor: pointer;
}
.sort-icon {
    font-size: 0.8em;
    margin-left: 3px;
}
.search-container {
    position: relative;
    display: flex;
    align-items: center;
}
.search-container input {
    padding-right: 25px;
}
.clear-search {
    position: absolute;
    right: 5px;
    background: none;
    border: none;
```

```
        font-size: 16px;
        cursor: pointer;
        padding: 0;
        width: 20px;
        height: 20px;
        display: flex;
        align-items: center;
        justify-content: center;
        opacity: 0.5;
    }
.clear-search:hover {
    opacity: 1;
}
.search-match {
    animation: pulse-highlight 2s ease-in-out 1;
}
.highlighted {
    background-color: var(--bg-highlight, #ffffa0);
    color: var(--text-color, #000);
    border-radius: 3px;
    padding: 0 2px;
}
.button-container {
    display: flex;
    gap: 5px;
}
.copy-value {
    opacity: 0.3;
    transition: opacity 0.2s;
    font-size: 0.8em;
    padding: 2px 5px;
    margin-left: 5px;
}
.value-cell:hover .copy-value {
    opacity: 1;
}
.address-cell {
    font-weight: bold;
}
.reference {
    color: var(--text-reference);
    text-decoration: underline;
    cursor: pointer;
}
.reference:focus {
    outline: 2px solid var(--border-focus);
}
.highlight-pulse {
    animation: pulse-highlight 2s ease-in-out 1;
}
.resolved-value {
    color: var(--text-resolved);
    margin-top: 3px;
    font-style: italic;
}
```

```
    max-height: 200px;
    overflow-y: auto;
}
/* Multi-line value display */
.multi-line-value {
    background-color: var(--bg-multi-line);
    padding: 5px;
    border-radius: 3px;
    border: 1px dashed var(--border-multi-line);
    max-height: 150px;
    overflow-y: auto;
}
/* Text editor */
.editor-container {
    display: flex;
    flex-direction: column;
    flex: 1;
    min-height: 200px;
    height: calc(100vh - 250px);
    min-height: 300px;
}
.editor-wrapper {
    display: flex;
    flex: 1;
    border: 1px solid #ccc;
    margin-top: 5px;
    height: 100%;
}
.line-numbers {
    display: flex;
    flex-direction: column;
    text-align: right;
    padding: 0 5px;
    background-color: rgba(0, 0, 0, 0.05);
    color: #888;
    font-family: monospace;
    border-right: 1px solid #ddd;
    user-select: none;
    overflow: hidden;
    font-size: 14px;
    line-height: 1.5;
}
.text-editor {
    width: 100%;
    flex: 1;
    padding: 10px;
    border: 1px solid var(--border-light);
    font-family: var(--font-mono);
    resize: none;
    tab-size: 4;
    -moz-tab-size: 4;
    min-height: 200px;
    overflow: auto;
    white-space: pre-wrap;
```

```
background-color: var(--bg-editor);
color: var(--text-primary);
font-size: 14px;
line-height: 1.5;
padding: 0 5px;
border: none;
}
.text-editor:focus {
outline: 2px solid var(--border-focus);
border-color: var(--border-focus);
}
.syntax-hints {
background-color: rgba(0,0,0,0.05);
padding: 8px;
border-radius: 4px;
font-size: 0.9em;
margin-bottom: 10px;
}
.syntax-hints code {
background-color: rgba(0,0,0,0.1);
padding: 2px 4px;
border-radius: 3px;
font-family: monospace;
}
/* Status bar */
.status-bar {
padding: 5px 10px;
background-color: var(--bg-section-title);
border-top: 1px solid var(--border-light);
font-size: 0.85rem;
color: var(--text-primary);
}
/* Toast notification */
.toast {
position: fixed;
top: 20px;
right: 20px;
padding: 10px 15px;
background-color: var(--bg-toast);
color: var(--text-contrast);
border-radius: 3px;
max-width: 300px;
z-index: 1000;
display: none;
box-shadow: var(--shadow-md);
}
.toast-fade-in {
animation: fade-in 0.3s ease-in-out;
}
.progress-toast {
position: fixed;
bottom: 20px;
right: 20px;
background-color: #333;
```

```
        color: white;
        padding: 15px;
        border-radius: 4px;
        z-index: 1000;
        box-shadow: 0 2px 10px rgba(0,0,0,0.2);
        display: flex;
        flex-direction: column;
        gap: 10px;
        min-width: 250px;
    }
.progress-bar {
    height: 10px;
    background-color: #555;
    border-radius: 5px;
    overflow: hidden;
}
.progress-fill {
    height: 100%;
    background-color: #4CAF50;
    transition: width 0.3s;
}
.progress-text {
    text-align: right;
    font-size: 0.9em;
    opacity: 0.8;
}
.extended {
    padding: 15px;
    max-width: 400px;
}
/* Modal styles */
.modal-overlay {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: rgba(0,0,0,0.5);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 100;
    display: none;
}
.modal {
    background-color: var(--bg-container);
    border-radius: 5px;
    padding: 20px;
    width: 90%;
    max-width: 350px;
    box-shadow: var(--shadow-lg);
}
.help-modal {
    max-width: 450px;
```

```
}

.modal-title {
    margin-bottom: 15px;
    font-size: 1.2rem;
    font-weight: bold;
    color: var(--text-primary);
}

.form-row {
    margin-bottom: 15px;
}

.form-row label {
    display: block;
    margin-bottom: 5px;
    color: var(--text-primary);
}

.form-row input {
    width: 100%;
    padding: 8px;
    border: 1px solid var(--border-light);
    border-radius: 3px;
    background-color: var(--bg-container);
    color: var(--text-primary);
}

.form-row input:focus {
    outline: 2px solid var(--border-focus);
    border-color: var(--border-focus);
}

.modal-buttons {
    display: flex;
    justify-content: flex-end;
    gap: 10px;
    margin-top: 20px;
}

.help-content {
    display: flex;
    flex-direction: column;
    gap: 10px;
    margin: 15px 0;
}

.shortcut-row {
    display: flex;
    justify-content: flex-start;
    align-items: center;
    gap: 15px;
}

.key {
    background-color: #eee;
    border: 1px solid #ddd;
    border-radius: 3px;
    padding: 2px 6px;
    font-family: monospace;
    font-size: 0.9em;
    box-shadow: 0 1px 1px rgba(0,0,0,0.2);
    min-width: 60px;
}
```

```
        text-align: center;
    }
/* Preserve whitespace styles */
.preserve-whitespace {
    white-space: pre-wrap;
    word-break: break-word;
    font-family: inherit;
}
.theme-cpp .reference {
    font-weight: bold;
}
.theme-cpp .resolved-value {
    font-family: var(--font-mono);
}
/* Animation keyframes */
@keyframes pulse-highlight {
    0%, 100% { background-color: transparent; }
    50% { background-color: var(--bg-highlight, rgba(255, 255, 0, 0.3)); }
}
@keyframes fade-in {
    0% { opacity: 0; transform: translateY(20px); }
    100% { opacity: 1; transform: translateY(0); }
}
/* Responsive adjustments */
@media (max-width: 768px) {
    .config-grid {
        grid-template-columns: 1fr;
    }

    .file-actions {
        flex-direction: column;
        align-items: flex-start;
    }

    .file-actions button {
        width: 100%;
        margin-top: 5px;
    }

    .bank-toolbar {
        flex-direction: column;
        align-items: flex-start;
        gap: 10px;
    }

    .bank-toolbar div:last-child {
        display: flex;
        gap: 5px;
    }

    .bank-toolbar div:last-child button {
        flex: 1;
    }
}
```

```
.addresses-table {
    font-size: 0.9rem;
}

.tab {
    padding: 5px 10px;
    font-size: 0.9rem;
}

/* Make sure all form inputs are large enough for touch on mobile */
button, input, select {
    min-height: 44px;
}

.theme-selector {
    position: relative;
    top: 0;
    right: 0;
    margin-bottom: 10px;
    justify-content: flex-end;
}

.toolbar-controls {
    width: 100%;
    flex-direction: column;
    align-items: stretch;
}

.search-container {
    width: 100%;
}

.search-container input {
    width: 100%;
}

}

/* For small screens, adapt table layout */
@media (max-width: 480px) {
    .addresses-table, .addresses-table tbody, .addresses-table tr {
        display: block;
        width: 100%;
    }

    .addresses-table th {
        display: none;
    }

    .addresses-table td {
        display: flex;
        padding: 8px;
        border-bottom: none;
    }

    .addresses-table td:first-child {
```

```
        font-weight: bold;
        background-color: var(--bg-section-title);
    }

    .addresses-table td:before {
        content: attr(data-label);
        font-weight: bold;
        flex: 0 0 80px;
    }

    .addresses-table tr {
        border: 1px solid var(--border-light);
        margin-bottom: 10px;
    }

/* Custom scrollbar for modern browsers */
::-webkit-scrollbar {
    width: 8px;
    height: 8px;
}

::-webkit-scrollbar-track {
    background: var(--bg-primary);
}

::-webkit-scrollbar-thumb {
    background: var(--border-medium);
    border-radius: 4px;
}

::-webkit-scrollbar-thumb:hover {
    background: var(--border-light);
}

};

doc.head.appendChild(styleEl);

// 2. Inject the application's HTML structure
body.innerHTML = `
<div class="theme-selector">
    <label for="themeSelect">Theme:</label>
    <select id="themeSelect">
        <option value="default">Default</option>
        <option value="cpp">C++ Developer</option>
        <option value="dark">Dark Mode</option>
        <option value="light">Light Mode</option>
    </select>
</div>
<div class="app-container">
    <header>
        <h1>DATA BANK MANAGER</h1>
        <p>Manage and query multi-level reference data files</p>
    </header>
    <div class="main-content">
        <section class="config-section">
            <h2>BANK CONFIGURATION</h2>
            <div class="config-grid">
```

```
        <div class="config-item">
            <label for="maxRegisters">Max Registers Per
        Bank</label>
            <input type="number" id="maxRegisters" min="1"
        max="100" value="10">
        </div>
        <div class="config-item">
            <label for="maxAddresses">Max Addresses Per
        Register</label>
            <input type="number" id="maxAddresses" min="1"
        max="100" value="20">
        </div>
        <div class="config-item">
            <label for="readingMode">Reading Mode</label>
            <select id="readingMode">
                <option value="raw">raw</option>
                <option value="resolve">resolve</option>
            </select>
        </div>
        </div>
        <div class="file-actions">
            <label>Bank Files</label>
            <button id="uploadButton" tabindex="0" aria-label="Upload
bank files">UPLOAD FILES</button>
            <button id="createNewButton" tabindex="0" aria-
label="Create a new bank">CREATE NEW BANK</button>
            <button id="downloadAllButton" tabindex="0" aria-
label="Download all banks as a ZIP file">DOWNLOAD ALL</button>
            <input type="file" id="fileUpload" multiple accept=".txt"
style="display: none;" aria-label="Upload bank files">
        </div>
    </section>
    <section class="banks-section">
        <h2>DATA BANKS</h2>
        <div class="tab-container" id="bankTabs" role="tablist">
            <div class="tab" id="addBankTab" role="tab" tabindex="0"
aria-selected="false">+ Add</div>
        </div>
        <div class="tab-contents" id="bankContents">
        </div>
    </section>
</div>
<div class="status-bar" id="statusBar" aria-live="polite">READY</div>
</div>
';

// 3. Adapt and execute the application's JavaScript logic

function loadScript(url, on_load) {
    const script = doc.createElement("script");
    script.src = url;
    script.onload = on_load;
    doc.head.appendChild(script);
}
```

```
const app = {
    maxRegisters: 10,
    maxAddresses: 20,
    readingMode: "raw",
    banks: {},
    currentBankId: null,
    editMode: {},
    currentTheme: "default",
    undoHistory: {},
    searchTerm: "",
    filterMode: "all"
};

const elements = {
    maxRegisters: doc.getElementById("maxRegisters"),
    maxAddresses: doc.getElementById("maxAddresses"),
    readingMode: doc.getElementById("readingMode"),
    uploadButton: doc.getElementById("uploadButton"),
    fileUpload: doc.getElementById("fileUpload"),
    createNewButton: doc.getElementById("createNewButton"),
    downloadAllButton: doc.getElementById("downloadAllButton"),
    bankTabs: doc.getElementById("bankTabs"),
    bankContents: doc.getElementById("bankContents"),
    statusBar: doc.getElementById("statusBar"),
    addBankTab: doc.getElementById("addBankTab"),
    themeSelect: doc.getElementById("themeSelect")
};

function initApp() {
    elements.maxRegisters.addEventListener("change", updateConfig);
    elements.maxAddresses.addEventListener("change", updateConfig);
    elements.readingMode.addEventListener("change", updateReadingMode);
    elements.uploadButton.addEventListener("click", () =>
elements.fileUpload.click());
    elements.fileUpload.addEventListener("change", handleFileUpload);
    elements.createNewButton.addEventListener("click", showNewBankModal);
    elements.downloadAllButton.addEventListener("click", downloadAllBanks);
    elements.addBankTab.addEventListener("click", showNewBankModal);
    elements.themeSelect.addEventListener("change", changeTheme);
    elements.addBankTab.addEventListener("keydown", (e) => {
        if (e.key === "Enter" || e.key === " ") {
            e.preventDefault();
            showNewBankModal();
        }
    });
    setupKeyboardShortcuts();
    loadConfigFromStorage();
    loadThemeFromStorage();
    showWelcomeMessage();
}

function showWelcomeMessage() {
    if (!localStorage.getItem("dataBankWelcomed")) {
```

```
const welcomeHtml = `

    <div style="margin-bottom: 10px"><strong>Welcome to Data Bank Manager!</strong></div>
        <div>Get started by:</div>
        <ul>
            <li>Uploading existing bank files using the "UPLOAD FILES" button.</li>
            <li>Creating a new bank using the "CREATE NEW BANK" button.</li>
        </ul>
        <div style="margin-top: 10px">Press F1 for keyboard shortcuts.
</div>
`;

GOS.ui.showDialog("Welcome!", welcomeHtml, { buttons: [ 'Got it!' ] });
localStorage.setItem("dataBankWelcomed", "true");
}

}

function setupKeyboardShortcuts() {
    doc.addEventListener("keydown", (e) => {
        if (e.target.tagName === "INPUT" || e.target.tagName === "TEXTAREA" || e.target.tagName === "SELECT") {
            return;
        }
        const modifierKey = e.ctrlKey || e.metaKey;
        if (modifierKey && e.key === "s" && app.currentBankId && app.editMode[app.currentBankId]) {
            e.preventDefault();
            saveBankEdits(app.currentBankId);
        }
        if (modifierKey && e.key === "z" && app.currentBankId && app.editMode[app.currentBankId]) {
            e.preventDefault();
            undoEdit(app.currentBankId);
        }
        if (modifierKey && e.key === "n") {
            e.preventDefault();
            showNewBankModal();
        }
        if (modifierKey && e.key === "d" && app.currentBankId) {
            e.preventDefault();
            downloadBank(app.currentBankId);
        }
        if (e.key === "F1") {
            e.preventDefault();
            showHelpOverlay();
        }
    });
}

function showHelpOverlay() {
    const helpContent =
        <div class="help-content" style="text-align: left;">
            <div class="shortcut-row"><span class="key">Ctrl+S</span>
<span>Save current bank (in edit mode)</span></div>
```

```
        <div class="shortcut-row"><span class="key">Ctrl+Z</span>
<span>Undo edit (in edit mode)</span></div>
        <div class="shortcut-row"><span class="key">Ctrl+N</span>
<span>Create a new bank</span></div>
        <div class="shortcut-row"><span class="key">Ctrl+D</span>
<span>Download the current bank</span></div>
        <div class="shortcut-row"><span class="key">F1</span> <span>Show
this help dialog</span></div>
        <div class="shortcut-row"><span class="key">Esc</span> <span>Close
dialogs</span></div>
    </div>`;
    GOS.ui.showDialog("Keyboard Shortcuts", helpContent, { buttons: ['Close'] });
};

function undoEdit(bankId) {
    if (app.undoHistory[bankId] && app.undoHistory[bankId].length > 0) {
        const previousState = app.undoHistory[bankId].pop();
        app.banks[bankId] = JSON.parse(previousState);
        const textEditor = doc.getElementById(`textEditor_${bankId}`);
        if (textEditor) {
            textEditor.value = generateBankText(app.banks[bankId]);
        }
        showToast("Undo successful");
    } else {
        showToast("Nothing to undo");
    }
}

function changeTheme() {
    const selectedTheme = elements.themeSelect.value;
    doc.body.classList.remove('theme-cpp', 'theme-dark', 'theme-light');
    if (selectedTheme !== 'default') {
        doc.body.classList.add(`theme-${selectedTheme}`);
    }
    app.currentTheme = selectedTheme;
    saveThemeToStorage();
    showToast(`Theme changed to ${selectedTheme}`);
}

function saveThemeToStorage() {
    try {
        localStorage.setItem("dataBankTheme", app.currentTheme);
    } catch (e) {
        console.warn("Could not save theme to local storage.", e);
    }
}

function loadThemeFromStorage() {
    try {
        const savedTheme = localStorage.getItem("dataBankTheme");
        if (savedTheme) {
            app.currentTheme = savedTheme;
            elements.themeSelect.value = savedTheme;
        }
    } catch (e) {
        console.error("Error loading theme from local storage.", e);
    }
}
```

```
        if (savedTheme !== 'default') {
            doc.body.classList.add(`theme-${savedTheme}`);
        }
    }
} catch (e) {
    console.warn("Could not load theme from local storage.", e);
}
}

function loadConfigFromStorage() {
try {
    const storedConfig = localStorage.getItem("dataBankConfig");
    if (storedConfig) {
        const config = JSON.parse(storedConfig);
        app.maxRegisters = config.maxRegisters || 10;
        app.maxAddresses = config.maxAddresses || 20;
        app.readingMode = config.readingMode || "raw";
        elements.maxRegisters.value = app.maxRegisters;
        elements.maxAddresses.value = app.maxAddresses;
        elements.readingMode.value = app.readingMode;
    }
} catch (e) {
    console.warn("Could not load config from local storage.", e);
}
}

function saveConfigToStorage() {
try {
    localStorage.setItem("dataBankConfig", JSON.stringify({
        maxRegisters: app.maxRegisters,
        maxAddresses: app.maxAddresses,
        readingMode: app.readingMode
    }));
} catch (e) {
    console.warn("Could not save config to local storage.", e);
}
}

function updateConfig() {
    app.maxRegisters = parseInt(elements.maxRegisters.value);
    app.maxAddresses = parseInt(elements.maxAddresses.value);
    saveConfigToStorage();
    showToast("Configuration updated");
}

function updateReadingMode() {
    app.readingMode = elements.readingMode.value;
    saveConfigToStorage();
    refreshAllBanks();
    showToast(`Reading mode changed to: ${app.readingMode}`);
}

function handleFileUpload(event) {
    const files = event.target.files;
```

```
if (!files || files.length === 0) return;
setStatusMessage(`Processing ${files.length} files...`);
let processedCount = 0;
let errorCount = 0;

const promises = Array.from(files).map((file) => {
    return new Promise((resolve) => {
        const match = file.name.match(/^(\d+)\.txt$/);
        if (!match) {
            showToast(`Skipped ${file.name}: Invalid filename format.`);
            errorCount++;
            resolve();
            return;
        }
        const bankId = parseInt(match[1]);
        const reader = new FileReader();
        reader.onload = function(e) {
            try {
                processFileContent(bankId, e.target.result);
                processedCount++;
            } catch (err) {
                errorCount++;
                showToast(`Failed to process ${file.name}: ${err.message}`);
            }
            resolve();
        };
        reader.onerror = function() {
            errorCount++;
            showToast(`Error reading file: ${file.name}`);
            resolve();
        };
        reader.readAsText(file);
    });
});

Promise.all(promises).then(() => {
    setStatusMessage("File processing complete.");
    if(errorCount > 0) {
        showToast(`Processed ${processedCount} files with ${errorCount} errors.`);
    } else {
        showToast(`Successfully processed ${processedCount} files.`);
    }
});
event.target.value = "";
}

function processFileContent(bankId, content) {
    const bank = parseBankData(bankId, content);
    app.banks[bankId] = bank;
    createBankTab(bankId);
    if (app.currentBankId === null) {
        selectBank(bankId);
```

```
        }

    }

    function parseBankData(bankId, content) {
        const lines = content.replace(/\r/g, '').split('\n');
        const bank = { id: bankId, registers: {} };
        let currentRegister = null;
        let inMultiLineValue = false;
        let currentAddressId = null;
        let multiLineBuffer = [];

        for (let i = 0; i < lines.length; i++) {
            const line = lines[i];
            if (inMultiLineValue) {
                if (line.trim() === '....') {
                    bank.registers[currentRegister].addresses[currentAddressId] =
multiLineBuffer.join('\n');
                    multiLineBuffer = [];
                    inMultiLineValue = false;
                } else {
                    multiLineBuffer.push(line);
                }
                continue;
            }

            if (!line.trim()) continue;

            if (!line.startsWith('\t')) {
                currentRegister = parseInt(line.trim());
                if (isNaN(currentRegister)) throw new Error(`Invalid register
number at line ${i + 1}`);
                if (!bank.registers[currentRegister]) {
                    bank.registers[currentRegister] = { addresses: {} };
                }
            } else {
                if (currentRegister === null) throw new Error(`Address found
before register at line ${i + 1}`);
                const addressLine = line.replace(/^\t/, '');
                const parts = addressLine.split('\t', 2);
                currentAddressId = parseInt(parts[0].trim());
                if (isNaN(currentAddressId)) throw new Error(`Invalid address ID
at line ${i + 1}`);

                if (parts.length === 1) {
                    bank.registers[currentRegister].addresses[currentAddressId] =
"";
                } else if (parts[1].trim() === '....') {
                    inMultiLineValue = true;
                    multiLineBuffer = [];
                } else {
                    bank.registers[currentRegister].addresses[currentAddressId] =
parts[1];
                }
            }
        }
    }
}
```

```
        }
        if (inMultiLineValue) throw new Error('Unclosed multi-line value at end of
file');
        return bank;
    }

function createBankTab(bankId) {
    if (doc.getElementById(`bankTab_${bankId}`)) return;
    const tab = doc.createElement("div");
    tab.className = "tab";
    tab.id = `bankTab_${bankId}`;
    tab.setAttribute("role", "tab");
    tab.tabIndex = 0;
    tab.innerHTML = `<span class="tab-title">Bank ${bankId}</span><button
class="tab-close" aria-label="Close Bank ${bankId}">&times;</button>`;

    tab.addEventListener("click", (e) => {
        if (!e.target.classList.contains("tab-close")) selectBank(bankId);
    });
    tab.addEventListener("keydown", (e) => {
        if (e.key === "Enter" || e.key === " ") {
            e.preventDefault();
            selectBank(bankId);
        }
    });
}

tab.querySelector(".tab-close").addEventListener("click", (e) => {
    e.stopPropagation();
    closeBank(bankId);
});

elements.bankTabs.insertBefore(tab, elements.addBankTab);
const content = doc.createElement("div");
content.className = "tab-content";
content.id = `bankContent_${bankId}`;
elements.bankContents.appendChild(content);
}

function closeBank(bankId) {
    GOS.ui.showDialog('Confirm Close', `Are you sure you want to close Bank
${bankId}? Any unsaved changes will be lost.`, { buttons: ['Cancel', 'Close'] })
    .then(result => {
        if (result.button === 'Close') {
            doc.getElementById(`bankTab_${bankId}`).remove();
            doc.getElementById(`bankContent_${bankId}`).remove();
            delete app.banks[bankId];
            delete app.editMode[bankId];
            delete app.undoHistory[bankId];

            if (app.currentBankId === bankId) {
                app.currentBankId = null;
                const bankIds = Object.keys(app.banks);
                if (bankIds.length > 0) {
                    selectBank(parseInt(bankIds[0]));
                }
            }
        }
    });
}
```

```
        }
    }
    showToast(`Bank ${bankId} closed.`);
}
});

function selectBank(bankId) {
    app.currentBankId = bankId;
    doc.querySelectorAll(".tab").forEach(tab => {
        tab.classList.remove("active");
        tab.setAttribute("aria-selected", "false");
    });
    const selectedTab = doc.getElementById(`bankTab_${bankId}`);
    if(selectedTab) {
        selectedTab.classList.add("active");
        selectedTab.setAttribute("aria-selected", "true");
    }
    doc.querySelectorAll(".tab-content").forEach(content =>
content.classList.remove("active"));
    const selectedContent = doc.getElementById(`bankContent_${bankId}`);
    if(selectedContent) {
        selectedContent.classList.add("active");
        displayBankData(bankId);
    }
}

function displayBankData(bankId) {
    const contentContainer = doc.getElementById(`bankContent_${bankId}`);
    if (!contentContainer) return;
    contentContainer.innerHTML = "";
    if (app.editMode[bankId]) {
        displayEditMode(bankId, contentContainer);
    } else {
        displayViewMode(bankId, contentContainer);
    }
}

function displayEditMode(bankId, container) {
    const toolbar = doc.createElement("div");
    toolbar.className = "bank-toolbar";
    toolbar.innerHTML =
        `

<span>Editing Bank ${bankId}</span><span
class="edit-hint">(Ctrl+S to save)</span></div>
        <div class="toolbar-actions">
            <button id="cancelEditBank_${bankId}">Cancel</button>
            <button id="saveEditBank_${bankId}" class="primary-
button">Save</button>
        </div>`;
    container.appendChild(toolbar);
    doc.getElementById(`cancelEditBank_${bankId}`).addEventListener("click", () => cancelBankEdits(bankId));
    doc.getElementById(`saveEditBank_${bankId}`).addEventListener("click", () => saveBankEdits(bankId));
}


```

```
const editorContainer = doc.createElement("div");
editorContainer.className = "editor-container";
const textEditor = doc.createElement("textarea");
textEditor.className = "text-editor";
textEditor.id = `textEditor_${bankId}`;
textEditor.value = generateBankText(app.banks[bankId]);
editorContainer.appendChild(textEditor);
container.appendChild(editorContainer);

textEditor.addEventListener("keydown", function(e) {
    if (e.key === "Tab") {
        e.preventDefault();
        const start = this.selectionStart;
        this.value = this.value.substring(0, start) + "\t" +
this.value.substring(this.selectionEnd);
        this.selectionStart = this.selectionEnd = start + 1;
    }
});

setTimeout(() => textEditor.focus(), 0);
}

function displayEditMode(bankId, container) {
    const bank = app.banks[bankId];
    container.innerHTML = `<div class="bank-toolbar"><div class="toolbar-
info"><h3 class="bank-title">Bank ${bankId}</h3></div><div class="toolbar-
controls"><button id="editBank_${bankId}">Edit</button><button
id="downloadBank_${bankId}">Download</button></div></div><div class="registers-
container"></div>`;
    doc.getElementById(`editBank_${bankId}`).addEventListener("click", () =>
startEditMode(bankId));
    doc.getElementById(`downloadBank_${bankId}`).addEventListener("click", () =>
downloadBank(bankId));
    const registersContainer = container.querySelector('.registers-
container');
    const registerIds = Object.keys(bank.registers).map(id =>
parseInt(id)).sort((a, b) => a - b);
    registerIds.forEach(registerId => {
        const register = bank.registers[registerId];
        const registerContainer = doc.createElement("div");
        registerContainer.className = "register-container";
        registerContainer.id = `bank_${bankId}_register_${registerId}`;
        registerContainer.innerHTML = `<div class="register-header"><div
class="register-title"><span class="collapse-icon">▼</span> Register ${registerId}
</div></div><div class="table-container"><table class="addresses-table"><thead>
<tr><th>Address</th><th>Value</th></tr></thead><tbody></tbody></table></div>`;
        const tbody = registerContainer.querySelector('tbody');
        const addressIds = Object.keys(register.addresses).map(id =>
parseInt(id)).sort((a, b) => a - b);
        addressIds.forEach(addressId => {
            const row = doc.createElement("tr");
            row.dataset.addressId = addressId; // Add data attribute for
precise selection
        });
    });
}
```

```
        const valueCell = doc.createElement("td");

valueCell.appendChild(formatValueWithReferences(register.addresses[addressId],
bankId, registerId, addressId));
    row.innerHTML = `<td>${addressId}</td>`;
    row.appendChild(valueCell);
    tbody.appendChild(row);
});
registerContainer.querySelector('.register-
header').addEventListener("click", () => {
    registerContainer.classList.toggle("collapsed");
    registerContainer.querySelector(".collapse-icon").textContent =
registerContainer.classList.contains("collapsed") ? "▶" : "▼";
});
registersContainer.appendChild(registerContainer);
});
}

function startEditMode(bankId) {
    app.undoHistory[bankId] = [JSON.stringify(app.banks[bankId])];
    app.editMode[bankId] = true;
    displayBankData(bankId);
}

function saveBankEdits(bankId) {
    const textEditor = doc.getElementById(`textEditor_${bankId}`);
    try {
        app.banks[bankId] = parseBankData(bankId, textEditor.value);
        app.editMode[bankId] = false;
        app.undoHistory[bankId] = [];
        displayBankData(bankId);
        showToast(`Bank ${bankId} saved.`);
    } catch (e) {
        GOS.ui.showDialog("Save Error", `Could not save Bank ${bankId}:<br>
<pre>${e.message}</pre>`, { buttons: ['OK'] });
    }
}

function cancelBankEdits(bankId) {
    app.editMode[bankId] = false;
    app.banks[bankId] = JSON.parse(app.undoHistory[bankId][0]);
    app.undoHistory[bankId] = [];
    displayBankData(bankId);
}

function generateBankText(bank) {
    let text = "";
    const registerIds = Object.keys(bank.registers).map(Number).sort((a, b) =>
a - b);
    for (const registerId of registerIds) {
        text += `${registerId}\n`;
        const addressIds =
Object.keys(bank.registers[registerId].addresses).map(Number).sort((a, b) => a -
b);
    }
}
```

```
        for (const addressId of addressIds) {
            const value = bank.registers[registerId].addresses[addressId] ||
            "";
            if (value.includes('\n')) {
                text += `\\t${addressId}\\t""\\n${value}\\n\\t""\\n`;
            } else {
                text += `\\t${addressId}\\t${value}\\n`;
            }
        }
        text += "\\n";
    }
    return text;
}

async function showNewBankModal() {
    const bankIds = Object.keys(app.banks).map(id => parseInt(id));
    const nextId = bankIds.length > 0 ? Math.max(...bankIds) + 1 : 1;
    const dialogBody = `
        <div class="form-row"><label for="newBankId">Bank ID Number</label>
<input type="number" id="newBankId" min="1" value="${nextId}"></div>
        <div class="form-row"><label for="newNumRegisters">Number of
Registers</label><input type="number" id="newNumRegisters" min="1" value="3">
</div>
        <div class="form-row"><label for="newNumAddresses">Addresses Per
Register</label><input type="number" id="newNumAddresses" min="1" value="5">
</div>`;

    // ☑ DEFINITIVE FIX: Use `window.parent.document` to find elements in
    // the main OS DOM.
    const getFormData = () => ({
        bankId: window.parent.document.getElementById('newBankId').value,
        numRegisters:
        window.parent.document.getElementById('newNumRegisters').value,
        numAddresses:
        window.parent.document.getElementById('newNumAddresses').value
    });

    const result = await GOS.ui.showDialog('Create New Bank', dialogBody, {
        buttons: ['Cancel', 'Create'], getFormData
    });

    if (result.button === 'Create' && result.formData) {
        createNewBankFromData(result.formData);
    }
}

function createNewBankFromData(data) {
    const bankId = parseInt(data.bankId);
    const numRegisters = parseInt(data.numRegisters);
    const numAddresses = parseInt(data.numAddresses);

    if (isNaN(bankId) || isNaN(numRegisters) || isNaN(numAddresses) || bankId
< 1 || numRegisters < 1 || numAddresses < 1) {
        return showToast("Invalid input. Please enter positive numbers.");
    }
}
```

```
if (app.banks[bankId]) {
    return showToast(`Bank ${bankId} already exists. Please choose a
different ID.`);
}

const bank = { id: bankId, registers: {} };
for (let r = 1; r <= numRegisters; r++) {
    bank.registers[r] = { addresses: {} };
    for (let a = 1; a <= numAddresses; a++) {
        bank.registers[r].addresses[a] = `Value ${r}.${a}`;
    }
}

app.banks[bankId] = bank;
createBankTab(bankId);
selectBank(bankId);
startEditMode(bankId);
showToast(`Bank ${bankId} created.`);
}

function downloadBank(bankId) {
    const bank = app.banks[bankId];
    if (!bank) return showToast(`Bank ${bankId} not found`);
    const content = generateBankText(bank);
    const blob = new Blob([content], { type: 'text/plain;charset=utf-8' });
    const link = window.parent.document.createElement('a');
    link.href = URL.createObjectURL(blob);
    link.download = `${bankId}.txt`;
    window.parent.document.body.appendChild(link);
    link.click();
    window.parent.document.body.removeChild(link);
    URL.revokeObjectURL(link.href);
}

function downloadAllBanks() {
    const bankIds = Object.keys(app.banks);
    if (bankIds.length === 0) return showToast("No banks to download");
    setStatusMessage("Creating ZIP file...");
    try {
        const zip = new JSZip();
        for (const bankId of bankIds) { zip.file(`${bankId}.txt`,
generateBankText(app.banks[bankId])); }
        zip.generateAsync({ type: "blob" }).then(function(content) {
            const link = window.parent.document.createElement("a");
            link.href = URL.createObjectURL(content);
            link.download = "databanks.zip";
            window.parent.document.body.appendChild(link);
            link.click();
            window.parent.document.body.removeChild(link);
            URL.revokeObjectURL(link.href);
            setStatusMessage("ZIP file downloaded.");
        });
    } catch(e) { showToast("Failed to create ZIP file."); console.error(e); }
}
```

```
function highlightReferenceInCurrentBank(registerId, addressId) {
    const registerContainer =
doc.getElementById(`bank_${app.currentBankId}_register_${registerId}`);
    if (registerContainer) {
        if (registerContainer.classList.contains("collapsed")) {
registerContainer.querySelector('.register-header').click(); }
        const rowToHighlight = registerContainer.querySelector(`tbody tr[data-
address-id="${addressId}"]`);
        if (rowToHighlight) {
            rowToHighlight.scrollIntoView({ behavior: 'smooth', block:
'center' });
            const originalColor = rowToHighlight.style.backgroundColor;
            rowToHighlight.style.transition = 'background-color 0.5s ease';
            rowToHighlight.style.backgroundColor = 'var(--bg-highlight)';
            setTimeout(() => {
                rowToHighlight.style.backgroundColor = originalColor;
                setTimeout(() =>
rowToHighlight.style.removeProperty('background-color'), 500);
            }, 2000);
            return;
        }
    }
    showToast(`Reference ${registerId}.${addressId} not found in current
view.`);
}

function handleReferenceClick(refPath) {
try {
    const [bankId, registerId, addressId] = refPath.split('.').map(part =>
parseInt(part));
    if (!app.banks[bankId]) return showToast(`Bank ${bankId} not found`);
    if (app.currentBankId !== bankId) {
        selectBank(bankId);
        setTimeout(() => highlightReferenceInCurrentBank(registerId,
addressId), 200);
    } else { highlightReferenceInCurrentBank(registerId, addressId); }
} catch (e) { showToast(`Invalid reference format: ${refPath}`); }
}

function resolveReference(refPath, visited = new Set()) {
if (visited.has(refPath)) {
    return `[Circular Reference: ${refPath}]`;
}
visited.add(refPath);

try {
    const [bankId, registerId, addressId] = refPath.split('.').map(part =>
parseInt(part));
    if (isNaN(bankId) || isNaN(registerId) || isNaN(addressId)) {
        return refPath; // Not a valid reference format
    }

    const bank = app.banks[bankId];
}
```

```
        if (!bank || !bank.registers[registerId] ||
bank.registers[registerId].addresses[addressId] === undefined) {
            return `[Unresolved: ${refPath}]`;
        }

        let value = bank.registers[registerId].addresses[addressId];

        // Recursively resolve nested references
        return value.replace(/(\d+\.\d+\.\d+)/g, (match) =>
resolveReference(match, new Set(visited)));

    } catch (e) {
        return `[Error resolving: ${refPath}]`;
    }
}

function formatValueWithReferences(value, bankId, registerId, addressId) {
    if (value === null || value === undefined) return doc.createTextNode("");

    const container = doc.createElement("div");
    container.style.whiteSpace = 'pre-wrap';
    if (value.includes('\n')) {
        container.classList.add("multi-line-value");
    }

    const refRegex = /(\d+\.\d+\.\d+)/g;

    // If in 'resolve' mode, resolve the entire value first
    if (app.readingMode === 'resolve') {
        let resolvedValue = value.replace(refRegex, (match) =>
resolveReference(match));

        const resolvedNode = doc.createElement('div');
        resolvedNode.className = 'resolved-value';
        resolvedNode.textContent = resolvedValue;

        // Show the original value with clickable references below the
        // resolved one
        const originalNode = doc.createElement('div');
        originalNode.className = 'original-reference';

        const parts = value.split(refRegex);
        parts.forEach((part, i) => {
            if (i % 2 === 0) {
                if (part) originalNode.appendChild(doc.createTextNode(part));
            } else {
                const refLink = doc.createElement("span");
                refLink.className = "reference";
                refLink.textContent = part;
                refLink.tabIndex = 0;
                refLink.addEventListener("click", () =>
handleReferenceClick(part));
                originalNode.appendChild(refLink);
            }
        })
    }
}
```

```
    });

    container.appendChild(resolvedNode);
    container.appendChild(originalNode);

} else { // 'raw' mode logic (original behavior)
    const parts = value.split(refRegex);
    parts.forEach((part, i) => {
        if (i % 2 === 0) {
            if (part) container.appendChild(doc.createTextNode(part));
        } else {
            const refLink = doc.createElement("span");
            refLink.className = "reference";
            refLink.textContent = part;
            refLink.tabIndex = 0;
            refLink.addEventListener("click", () =>
handleReferenceClick(part));
            container.appendChild(refLink);
        }
    });
}

return container;
}

function refreshAllBanks() {
    if (app.currentBankId !== null) {
        displayBankData(app.currentBankId);
    }
}

function setStatusMessage(message) {
    if (elements.statusBar) elements.statusBar.textContent = message;
}

function showToast(message) {
    GOS.ui.showNotification("Data Bank", message, 3000);
}

function escapeHtml(text) {
    const div = doc.createElement('div');
    div.textContent = text;
    return div.innerHTML;
}

// Load JSZip library and then initialize the app
loadScript("https://cdnjs.cloudflare.com/ajax/libs/jszip/3.10.1/jszip.min.js",
() => {
    initApp();
});
}
```

## [23] GenesisOS/databank-manager.json

- **Bytes:** 463
- **Type:** text

```
{  
  "id": "databank-manager",  
  "title": "Data Bank Manager",  
  "description": "A tool to manage and query multi-level, cross-referenced data files, suitable for complex configurations or structured data analysis.",  
  "icon": "\ud83d\uddc4\ufe0f",  
  "category": "Utilities",  
  "version": "1.0.0",  
  "entry": "databank-manager.js",  
  "permissions": [],  
  "window": {  
    "width": 1200,  
    "height": 800,  
    "resizable": true  
  }  
}
```

## [24] GenesisOS/db-json-repl.js

- **Bytes:** 14788
- **Type:** text

```
export function initialize(gosApiProxy) {  
  const container = gosApiProxy.window.getContainer();  
  const doc = container.ownerDocument;  
  
  // --- Styles (minimal rectangle center) ---  
  const style = doc.createElement('style');  
  style.textContent = `  
    :root {  
      --bg-color: #1e1e2e;  
      --editor-bg: #282a36;  
      --text-color: #cdd6f4;  
      --line-num-color: #6c7086;  
      --border-color: #313244;  
      --focus-color: #f5c2e7;  
      --placeholder-color: rgba(205,214,244,.5);  
      --scrollbar-bg: #313244;  
      --scrollbar-thumb: #585b70;  
      --scrollbar-thumb-hover: #6c7086;  
      --err: #e74c3c;  
    }  
    .gos-root {  
      font-family: 'Fira Code', 'Cascadia Code', 'JetBrains Mono', monospace;  
      background: var(--bg-color);  
    }  
  `};  
  style.innerHTML = `  
    :root {  
      --bg-color: #1e1e2e;  
      --editor-bg: #282a36;  
      --text-color: #cdd6f4;  
      --line-num-color: #6c7086;  
      --border-color: #313244;  
      --focus-color: #f5c2e7;  
      --placeholder-color: rgba(205,214,244,.5);  
      --scrollbar-bg: #313244;  
      --scrollbar-thumb: #585b70;  
      --scrollbar-thumb-hover: #6c7086;  
      --err: #e74c3c;  
    }  
    .gos-root {  
      font-family: 'Fira Code', 'Cascadia Code', 'JetBrains Mono', monospace;  
      background: var(--bg-color);  
    }  
  `;  
  container.appendChild(style);  
  gosApiProxy.setRootElement(style);  
}  
initialize();
```

```
min-height: 100%;  
display:flex;align-items:center;justify-content:center;padding:20px;  
}  
.editor-container {  
width: 90%; max-width: 800px; height: 70vh;  
background: var(--editor-bg);  
border: 2px solid var(--border-color); border-radius: 8px;  
box-shadow: 0 4px 30px rgba(0,0,0,.3);  
transition: border-color .3s ease, box-shadow .3s ease;  
display:flex; overflow:hidden;  
}  
.editor-container:focus-within {  
outline:none; border-color: var(--focus-color);  
box-shadow: 0 0 0 3px rgba(245,194,231,.3), 0 4px 30px rgba(0,0,0,.3);  
}  
.editor-container.dup { border-color: var(--err); box-shadow: 0 0 0 3px  
rgba(231,76,60,.3), 0 4px 30px rgba(0,0,0,.3); }  
#line-numbers {  
background: var(--editor-bg); color: var(--line-num-color);  
padding: 15px 10px 15px 15px; font-size: 16px; line-height: 1.6;  
user-select:none; text-align:right; overflow-y:hidden; min-width:45px;  
border-right: 1px solid var(--border-color);  

```

```
container.innerHTML = '';
container.appendChild(root);

const editor = root.querySelector('#editor');
// --- Safe confirm (works in sandbox without allow-modals) ---
const confirmAsync = (message) => {
  try {
    if (gosApiProxy && gosApiProxy.ui && typeof gosApiProxy.ui.confirm ===
'function') {
      return gosApiProxy.ui.confirm(message);
    }
  } catch(_) {}
  // DOM-based fallback modal
  return new Promise((resolve) => {
    const doc = gosApiProxy.window.getContainer().ownerDocument;
    const overlay = doc.createElement('div');
    overlay.style.position = 'fixed';
    overlay.style.inset = '0';
    overlay.style.background = 'rgba(0,0,0,.55)';
    overlay.style.display = 'grid';
    overlay.style.placeItems = 'center';
    overlay.style.zIndex = '99999';
    const box = doc.createElement('div');
    box.style maxWidth = '420px';
    box.style.width = '90%';
    box.style.background = '#282a36';
    box.style.color = '#cdd6f4';
    box.style.border = '1px solid #313244';
    box.style.borderRadius = '10px';
    box.style.boxShadow = '0 10px 30px rgba(0,0,0,.35)';
    box.style.padding = '16px';
    box.innerHTML = `<div style="margin-bottom:12px;font-family:'Fira
Code','Cascadia Code','JetBrains Mono',monospace;">${message}</div>`;
    const row = doc.createElement('div');
    row.style.display = 'flex';
    row.style.gap = '8px';
    row.style.justifyContent = 'flex-end';
    const btn = (label, ok) => {
      const b = doc.createElement('button');
      b.textContent = label;
      b.style.background = '#202124';
      b.style.color = '#f1f1f1';
      b.style.border = '1px solid #313244';
      b.style.padding = '8px 12px';
      b.style.borderRadius = '8px';
      b.style.cursor = 'pointer';
      b.onclick = () => { overlay.remove(); resolve(ok); };
      return b;
    };
    row.appendChild(btn('Cancel', false));
    row.appendChild(btn('OK', true));
    box.appendChild(row);
    overlay.appendChild(box);
    doc.body.appendChild(overlay);
  });
}
```

```
    });

const ta = root.querySelector('#prompt-text');
const ln = root.querySelector('#line-numbers');

// --- DB state & helpers ---
const storeKey = 'db-json.json'; // same name for compatibility with your web
version
let DB = { items: [], dupsBlocked: 0, lastSaved: 'never' };
let page = 0; const pageSize = 5;
let anchor = 0; // prompt-lock boundary
let prevValue = '';

const nowStr = () => new Date().toLocaleString();
const sanitize = s => (s||'').replace(/[\s\n\t]+/g,'')
  .trim().replace(/[^.]+\s*$/,''); // strip trailing dots
const norm = s => sanitize(s).toLowerCase();

const save = () => { localStorage.setItem(storeKey, JSON.stringify(DB));
DB.lastSaved = nowStr(); };
const load = () => { const raw = localStorage.getItem(storeKey); if(!raw) return
false; try{ const o=JSON.parse(raw); if(Array.isArray(o.items)) DB={
dupsBlocked:0, lastSaved:'never', ...o }; return true; }catch(e){ return false; }};

const isDup = (s, exceptId=null) => { const n=norm(s); return !!n &&
DB.items.some(it=> it.id!==exceptId && norm(it.text)===n ); };
const add = (text) => { const v=sanitize(text); if(!v) return
{ok:false,reason:'empty'}; if(isDup(v)){ DB.dupsBlocked++; return
{ok:false,reason:'dup'}; } DB.items.push({ id: crypto.randomUUID(), text:v,
createdAt:new Date().toISOString(), updatedAt:new Date().toISOString() }); save();
return {ok:true}; };
const edit = (id, text) => { const v=sanitize(text); if(!v) return false;
if(isDup(v, id)) return 'dup'; const it=DB.items.find(x=>x.id==id); if(!it)
return false; it.text=v; it.updatedAt=new Date().toISOString(); save(); return
true; };
const delById = (id) => { const n=DB.items.length; DB.items =
DB.items.filter(x=>x.id!==id); if(DB.items.length!==n){ save(); return true; }
return false; };
const delByText = (t) => { const n=norm(t); const m=DB.items.length; DB.items =
DB.items.filter(x=>norm(x.text)!==n); if(DB.items.length!==m){ save(); return
true; } return false; };

const toLearn = () => DB.items.map(it=>it.text).join(' . ') + (DB.items.length?
'.:' );
const toJson = () => JSON.stringify(DB, null, 2);

const updateLineNumbers = () => { const lines = ta.value.split('\n').length;
ln.textContent = Array.from({length:lines}, (_,i)=>String(i+1)).join('\n');
ln.scrollTop = ta.scrollTop; };
const setCaretToEnd = () => { ta.selectionStart = ta.selectionEnd =
ta.value.length; };
```

```

const print = (s='') => {
  const needsNL = ta.value && !ta.value.endsWith('\n');
  ta.value += (needsNL ? '\n' : '') + s;
  ta.value += (s && !s.endsWith('\n')) ? '\n' : '';
  anchor = ta.value.length;
  prevValue = ta.value;
  setCaretToEnd(); ta.scrollTop = ta.scrollHeight; updateLineNumbers();
};

const printListPage = () => {
  const total = DB.items.length; const pages = Math.max(1, Math.ceil(total / pageSize));
  if(page >= pages) page = pages - 1; if(page < 0) page = 0;
  const start = page * pageSize; const end = Math.min(total, start + pageSize);
  print(`LIST ${start+1}-${end} of ${total}) • :next :prev`);
  for(let i=start;i<end;i++){
    const it = DB.items[i];
    print(` - ${String(i+1).padStart(3, ' ')} | ${it.text}
[${it.id.slice(0,8)}]`);
  }
};

const onTyping = () => {
  const current = ta.value.slice(anchor);
  if(current.startsWith(':')){ editor.classList.remove('dup'); return; }
  editor.classList.toggle('dup', isDup(current));
};

const runLine = (line) => {
  const raw = line.trim(); if(!raw) return;
  if(!raw.startsWith(':')){
    const res = add(raw);
    if(res.ok) print('✓ added'); else if(res.reason==='dup') print('X
duplicate blocked'); else print('. empty ignored');
    return;
  }
  const [cmd, ...rest] = raw.split(/\s+/); const args = rest.join(' ');
  switch(cmd){
    case ':help':
      print(`OVERVIEW
• Plain text adds an entry (duplicate-safe).
• Commands start with ':'`)
```

## COMMANDS

:help	Show this help.
:list	Show up to 5 entries (use :next / :prev).
:next / :prev	Page list forward/back.
:search <q>	Search entries.
:add <text>	Add entry.
:edit <id> <text>	Edit by id.
:del <id text>	Delete by id or exact text.
:clear	Remove ALL entries.
:stats	Show counts & last-saved.
:export json learn	Download db-json.json or LEARN.txt.

```

:import json|learn      Import from a file chooser.
:save / :load          Persist via localStorage.`);
    break;
  case ':list': page = 0; printListPage(); break;
  case ':next': page++; printListPage(); break;
  case ':prev': page--; printListPage(); break;
  case ':search': {
    const q = norm(args); const hits =
DB.items.filter(it=>norm(it.text).includes(q));
    print(`SEARCH '${args}' → ${hits.length} hit(s)`);
    hits.slice(0, pageSize).forEach((it,i)=> print(` -
${String(i+1).padStart(3, ' ')} | ${it.text} [${it.id.slice(0,8)}]`));
    if(hits.length>pageSize) print(` ... ${hits.length - pageSize} more (refine
search)`);
  } break;
  case ':add': { const res = add(args); print(res.ok? '✓ added' :
(res.reason==='dup'? '✗ duplicate blocked' : '· empty ignored')); } break;
  case ':del': { const tok=args.trim(); if(!tok){ print('usage: :del
<id|text>'); break; } const ok = tok.length>=8 &&
DB.items.some(it=>it.id.startsWith(tok)) ?
delById(DB.items.find(it=>it.id.startsWith(tok)).id) : delByText(tok); print(ok?
'✓ deleted' : 'not found'); } break;
  case ':edit': { const m=args.match(/^\s+(\S+)\s+([\s\S]+)$/); if(!m){
print('usage: :edit <id> <text>'); break; } const r=edit(m[1], m[2]);
print(r==true? '✓ updated' : (r==='dup'? '✗ duplicate blocked' : 'not
found')); } break;
  case ':clear':
    confirmAsync('Clear all entries?').then(ok=>{
      if(ok){ DB.items=[]; save(); print('✓ cleared'); }
    });
  break;
  case ':stats': print(`count ${DB.items.length} • duplicates blocked
${DB.dupsBlocked} • last saved ${DB.lastSaved}`); break;
  case ':export': { const t=args.toLowerCase(); if(t==='json') download('db-
json.json', toJson(), 'application/json'); else if(t==='learn')
download('LEARN.txt', toLearn(), 'text/plain'); else print('usage: :export
json|learn'); } break;
  case ':import': { const t=args.toLowerCase(); if(t==='json')
pickFile('application/json,.json', importJson); else if(t==='learn')
pickFile('.txt', importLearn); else print('usage: :import json|learn'); } break;
  case ':save': save(); print('✓ saved'); break;
  case ':load': print(load()? '✓ loaded' : 'nothing to load'); break;
  default: print(`unknown command: ${cmd}. Type :help`);
}
};

// downloads & import helpers
const download = (name, content, mime='text/plain') => {
  const blob = new Blob([content],{type:mime});
  const a = doc.createElement('a'); a.href = URL.createObjectURL(blob);
a.download = name;
  doc.body.appendChild(a); a.click(); setTimeout(()=>{
URL.revokeObjectURL(a.href); a.remove(); },0);
};

```

```

const pickFile = (accept, cb) => { const inp=doc.createElement('input');
inp.type='file'; inp.accept=accept; inp.onchange=async(e)=>{ const
f=e.target.files[0]; if(!f) return; const txt=await f.text(); cb(txt); };
inp.click(); };

const importLearn = (txt) => { const parts =
txt.replace(/\r/g, '').split('.').map(s=>sanitize(s)).filter(Boolean); let adds=0,
dups=0; parts.forEach(p=>{ if(isDup(p)) dups++; else { DB.items.push({ id:
crypto.randomUUID(), text:p, createdAt:new Date().toISOString(), updatedAt:new
Date().toISOString() }); adds++; }}); DB.dupsBlocked += dups; save();
print(`Imported ${adds} entr${adds==1?'y':'ies'}${dups?` , ${dups} duplicate
skipped`:''}).`);

const importJson = (txt) => { try{ const o=JSON.parse(txt);
if(!Array.isArray(o.items)) throw new Error('Invalid JSON'); let adds=0, dups=0;
o.items.forEach(it=>{ const v=sanitize(it.text); if(!v) return; if(isDup(v))
dups++; else { DB.items.push({ id: crypto.randomUUID(), text:v, createdAt:new
Date().toISOString(), updatedAt:new Date().toISOString() }); adds++; }});
DB.dupsBlocked += dups; save(); print(`Merged ${adds}
entr${adds==1?'y':'ies'}${dups?` , ${dups} duplicate skipped`:'}`).}catch(e){
print('Import failed: '+e.message); } };

// guards to protect printed output (prompt-lock)
ta.addEventListener('beforeinput', (e)=>{
  const start = ta.selectionStart, end = ta.selectionEnd;
  const isDeletion = e.inputType?.startsWith('delete');
  if(start < anchor || (isDeletion && end <= anchor)) { e.preventDefault();
setCaretToEnd(); }
});

ta.addEventListener('keydown', (e)=>{
  if(e.key==='Enter' && !e.shiftKey){ e.preventDefault(); const line =
ta.value.slice(anchor); runLine(line); print(''); return; }
  if(e.key==='Home'){ e.preventDefault(); ta.selectionStart = ta.selectionEnd =
anchor; return; }
  if(e.key==='Backspace' && ta.selectionStart<=anchor &&
ta.selectionEnd<=anchor){ e.preventDefault(); return; }
});

ta.addEventListener('input', ()=>{
  if(ta.selectionStart < anchor || ta.value.length < anchor){
    const keep = prevValue.slice(0, anchor);
    const tail = ta.value.slice(Math.max(anchor, 0));
    ta.value = keep + tail; setCaretToEnd();
  }
  prevValue = ta.value; onTyping(); updateLineNumbers();
});

ta.addEventListener('scroll', ()=>{ ln.scrollTop = ta.scrollTop; });

// boot
load();
print('Ready. Type :help for documentation.');
}

```

- **Bytes:** 426
- **Type:** text

```
{
  "id": "db-json-repl",
  "title": "DB\u2011JSON REPL",
  "description": "Minimal, duplicate\u2011safe LEARN.txt & db\u2011json.json editor with REPL shell.",
  "version": "1.0.0",
  "author": "You",
  "icon": "\ud83d\uddc2\ufe0f",
  "category": "Development",
  "entry": "db-json-repl.js",
  "permissions": [],
  "window": {
    "width": 900,
    "height": 600,
    "resizable": true
  }
}
```

## [26] GenesisOS/digital-linguist.js

- **Bytes:** 25323
- **Type:** text

```
export function initialize(gosApiProxy) {
  const container = gosApiProxy.window.getContainer();
  const doc = container.ownerDocument;

  // 1. Inject CSS Styles
  const style = doc.createElement('style');
  style.textContent =
    ':root {
      --primary-color: #007bff; --secondary-color: #6c757d; --bg-color: #f8f9fa;
      --surface-color: #ffffff; --border-color: #dee2e6; --text-color: #212529;
      --header-color: #343a40; --pre-bg-color: #e9ecef; --success-color: #28a745;
    }
    body {
      font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvetica, Arial, sans-serif;
      background-color: var(--bg-color); color: var(--text-color); margin: 0; padding: 1rem;
      height: 100vh; box-sizing: border-box;
    }
    .workstation-grid {
      display: grid; grid-template-columns: 400px 1fr; gap: 1.5rem; max-
```

```
width: 1400px; margin: auto; height: 100%;  
}  
.panel {  
    background-color: var(--surface-color); border: 1px solid var(--  
border-color);  
    border-radius: 8px; box-shadow: 0 4px 12px rgba(0,0,0,0.05); padding:  
1.5rem;  
    display: flex; flex-direction: column;  
}  
#controls-panel {  
    overflow-y: auto;  
}  
h1, h2, h3 { color: var(--header-color); border-bottom: 2px solid var(--  
border-color); padding-bottom: 0.5rem; margin-top: 0; }  

```

```
// 2. Inject HTML Structure
container.style.height = '100%';
container.style.overflow = 'hidden';
container.innerHTML =
    <div class="workstation-grid">
        <div id="controls-panel" class="panel">
            <h2>System Network Controls</h2>
            <div class="control-group">
                <label for="system-select">Managed Systems</label>
                <select id="system-select"></select>
                <div class="button-group">
                    <button id="new-system-btn" class="secondary-button">New</button>
                    <button id="delete-system-btn" class="secondary-button">Delete</button>
                </div>
            </div>
            <div class="control-group">
                <label for="system-name">System Name</label>
                <input type="text" id="system-name" value="Default System">
            </div>

            <h3>Geometric Ratio (1:d3)</h3>
            <div class="control-group">
                <label for="d_value">Cube Dimension ('d')</label>
                <input type="number" id="d_value" value="2" min="2" step="1">
                <div id="ratio-display-output" class="ratio-display">Target
Ratio: 1/9 (11.11%)</div>
            </div>

            <h3>Genetic Algorithm Parameters</h3>
            <div class="control-group">
                <label for="permLength">Permutation Length</label>
                <input type="number" id="permLength" value="50" min="10"
max="200">
            </div>
            <div class="control-group">
                <label for="populationSize">Population Size</label>
                <input type="number" id="populationSize" value="100" min="10"
max="500">
            </div>
            <div class="control-group">
                <label for="numGenerations">Number of Generations</label>
                <input type="number" id="numGenerations" value="100" min="10"
max="1000">
            </div>

            <h3>Alphabet Configuration</h3>
            <div class="control-group">
                <label for="alphabet-g">Target Alphabet (Component G)</label>
                <textarea id="alphabet-g">abcdefghijklm</textarea>
            </div>

            <div class="composition-box">
```

```
<h3>System Composition</h3>
<div class="control-group">
    <label for="parent-a-select">Parent System A</label>
    <select id="parent-a-select"></select>
</div>
<div class="control-group">
    <label for="parent-b-select">Parent System B</label>
    <select id="parent-b-select"></select>
</div>
<div class="control-group">
    <label for="alphabet-op-select">Alphabet
Composition</label>
    <select id="alphabet-op-select">
        <option value="union">Union (A ∪ B)</option>
        <option value="intersection">Intersection (A ∩ B)
    </option>
    </select>
</div>
    <button id="compose-btn" class="success-button">Compose New
System</button>
</div>
<h3 style="margin-top: 2rem;">Execution & File I/O</h3>
<button id="run-btn">▶ Run Active System</button>
<div class="button-group">
    <button id="save-btn" class="secondary-button">Save
Session</button>
    <button id="load-btn" class="secondary-button">Load
Session</button>
</div>
<input type="file" id="file-loader" class="file-input"
accept=".json">
</div>

<div id="results-panel" class="panel">
    <h2>Results & Analysis</h2>
    <div id="status">Status: Idle. Configure a system and press 'Run'.
</div>
    <h3>Best Found Permutation:</h3>
    <pre id="output">---</pre>
    <h3>Best Fitness Score (Closer to 0 is better):</h3>
    <pre id="fitness">---</pre>
    <h3>Final Ratio of Target Characters:</h3>
    <pre id="ratio">---</pre>
</div>
</div>
`;

// 3. Application Logic
class GAWorkstation {
    constructor() {
        this.DOM = {
            systemSelect: doc.getElementById('system-select'),
            newSystemBtn: doc.getElementById('new-system-btn'),
            deleteSystemBtn: doc.getElementById('delete-system-btn'),

```

```
        systemName: doc.getElementById('system-name'),
        d_value: doc.getElementById('d_value'),
        ratioDisplay: doc.getElementById('ratio-display-output'),
        permLength: doc.getElementById('permLength'),
        populationSize: doc.getElementById('populationSize'),
        numGenerations: doc.getElementById('numGenerations'),
        alphabetG: doc.getElementById('alphabet-g'),
        parentASelect: doc.getElementById('parent-a-select'),
        parentBSelect: doc.getElementById('parent-b-select'),
        alphabetOpSelect: doc.getElementById('alphabet-op-select'),
        composeBtn: doc.getElementById('compose-btn'),
        runBtn: doc.getElementById('run-btn'),
        saveBtn: doc.getElementById('save-btn'),
        loadBtn: doc.getElementById('load-btn'),
        fileLoader: doc.getElementById('file-loader'),
        status: doc.getElementById('status'),
        output: doc.getElementById('output'),
        fitness: doc.getElementById('fitness'),
        ratio: doc.getElementById('ratio')
    };

    this.FULL_ALPHABET =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 _'.split('');
    this.state = {
        managedSystems: {},
        activeSystemId: null
    };

    this.init();
}

init() {
    this.createNewSystem({ name: "Default System d=2" });
    this.bindEventListeners();
}

bindEventListeners() {
    this.DOM.newSystemBtn.addEventListener('click', () =>
this.createNewSystem({ name: "New System" }));
    this.DOM.deleteSystemBtn.addEventListener('click', () =>
this.deleteActiveSystem());
    this.DOM.composeBtn.addEventListener('click', () =>
this.composeSystem());
    this.DOM.systemSelect.addEventListener('change', (e) =>
this.setActiveSystem(e.target.value));
    this.DOM.d_value.addEventListener('input', () =>
this.updateRatioDisplay());

    this.DOM.runBtn.addEventListener('click', () => this.runEvolution());
    this.DOM.saveBtn.addEventListener('click', () => this.saveSession());
    this.DOM.loadBtn.addEventListener('click', () =>
this.DOM.fileLoader.click());
    this.DOM.fileLoader.addEventListener('change', (e) =>
this.loadSession(e));
}
```

```
        const inputs = doc.querySelectorAll('#controls-panel input, #controls-panel textarea');
        inputs.forEach(input => {
            input.addEventListener('change', () => this.updateSystemFromUI());
        });
    }

    createNewSystem(options = {}) {
        const newId = `system_${Date.now()}`;
        const newSystem = {
            id: newId,
            name: options.name || "Composed System",
            createdAt: new Date().toISOString(),
            config: options.config || {
                d_value: 2,
                permLength: 50,
                populationSize: 100,
                numGenerations: 100,
                mutationRate: 0.02,
                elitismRate: 0.1,
                alphabetG: "abcdefghijklmnopqrstuvwxyz"
            },
            results: null
        };
        this.state.managedSystems[newId] = newSystem;
        this.updateSystemSelects();
        this.setActiveSystem(newId);
    }

    setActiveSystem(id) {
        if (!this.state.managedSystems[id]) return;
        this.state.activeSystemId = id;
        this.updateUIFromState();
        this.DOM.systemSelect.value = id;
    }

    deleteActiveSystem() {
        if (Object.keys(this.state.managedSystems).length <= 1) {
            gosApiProxy.ui.showNotification('Action Denied', 'Cannot delete the last system.', 3000);
            return;
        }
        delete this.state.managedSystems[this.state.activeSystemId];
        this.updateSystemSelects();
        const newActiveId = Object.keys(this.state.managedSystems)[0];
        this.setActiveSystem(newActiveId);
    }

    updateSystemSelects() {
        const selects = [this.DOM.systemSelect, this.DOM.parentASelect,
        this.DOM.parentBSelect];
        const currentSystemValue = this.DOM.systemSelect.value;
        selects.forEach(sel => sel.innerHTML = '');
    }
}
```

```
        for (const id in this.state.managedSystems) {
            const system = this.state.managedSystems[id];
            const option = doc.createElement('option');
            option.value = id;
            option.textContent = system.name;
            selects.forEach(sel => sel.appendChild(option.cloneNode(true)));
        }

        this.DOM.systemSelect.value = currentSystemValue;
        if (this.DOM.parentASelect.options.length > 0)
    this.DOM.parentASelect.selectedIndex = 0;
        if (this.DOM.parentBSelect.options.length > 1)
    this.DOM.parentBSelect.selectedIndex = 1;
    }

    updateUIFromState() {
        const system = this.state.managedSystems[this.state.activeSystemId];
        if (!system) return;

        this.DOM.systemName.value = system.name;
        this.DOM.d_value.value = system.config.d_value;
        this.DOM.permLength.value = system.config.permLength;
        this.DOM.populationSize.value = system.config.populationSize;
        this.DOM.numGenerations.value = system.config.numGenerations;
        this.DOM.alphabetG.value = system.config.alphabetG;
        this.updateRatioDisplay();

        if (system.results) {
            this.DOM.output.textContent = system.results.bestPermutation;
            this.DOM.fitness.textContent =
        system.results.bestFitness.toExponential(4);
            this.DOM.ratio.textContent =
` ${system.results.finalRatio.toFixed(4)} (Target:
${system.results.targetRatio.toFixed(4)})`;
        } else {
            this.DOM.output.textContent = "---";
            this.DOM.fitness.textContent = "---";
            this.DOM.ratio.textContent = "---";
        }
    }

    updateSystemFromUI() {
        const system = this.state.managedSystems[this.state.activeSystemId];
        if (!system) return;

        system.name = this.DOM.systemName.value;
        system.config.d_value = parseInt(this.DOM.d_value.value, 10);
        system.config.permLength = parseInt(this.DOM.permLength.value, 10);
        system.config.populationSize = parseInt(this.DOM.populationSize.value,
10);
        system.config.numGenerations = parseInt(this.DOM.numGenerations.value,
10);
        system.config.alphabetG = this.DOM.alphabetG.value;
    }
}
```

```
        this.updateSystemSelects();
        this.updateRatioDisplay();
    }

    updateRatioDisplay() {
        const d = parseInt(this.DOM.d_value.value, 10);
        if (isNaN(d) || d < 2) {
            this.DOM.ratioDisplay.textContent = "Invalid 'd' value.";
            return;
        }
        const denominator = 1 + Math.pow(d, 3);
        const ratio = 1 / denominator;
        this.DOM.ratioDisplay.textContent = `Target Ratio: 1/${denominator}
(${(ratio * 100).toFixed(2)})`;
    }

    composeSystem() {
        const parentAId = this.DOM.parentASelect.value;
        const parentBId = this.DOM.parentBSelect.value;

        if (parentAId === parentBId) {
            gosApiProxy.ui.showNotification('Error', 'Please select two
different parent systems.', 4000);
            return;
        }

        const parentA = this.state.managedSystems[parentAId];
        const parentB = this.state.managedSystems[parentBId];

        const newConfig = {
            d_value: Math.round((parentA.config.d_value +
parentB.config.d_value) / 2),
            permLength: Math.round((parentA.config.permLength +
parentB.config.permLength) / 2),
            populationSize: Math.round((parentA.config.populationSize +
parentB.config.populationSize) / 2),
            numGenerations: Math.round((parentA.config.numGenerations +
parentB.config.numGenerations) / 2),
            mutationRate: 0.02,
            elitismRate: 0.1,
            alphabetG: ""
        };
        const setA = new Set(parentA.config.alphabetG.split(''));
        const setB = new Set(parentB.config.alphabetG.split(''));
        const op = this.DOM.alphabetOpSelect.value;
        let resultSet = (op === 'union') ? new Set([...setA, ...setB]) : new
Set([...setA].filter(x => setB.has(x)));
        newConfig.alphabetG = [...resultSet].sort().join('');

        const newName = `Composition of ${parentA.name} & ${parentB.name}`;
        this.createNewSystem({ name: newName, config: newConfig });
    }
}
```

```
saveSession() {
    const system = this.state.managedSystems[this.state.activeSystemId];
    if (!system) return;
    const sessionData = JSON.stringify(system, null, 2);
    const schemaData = JSON.stringify(this.generateSchema(system), null,
2);
    this.downloadFile(sessionData, `${system.name.replace(/\s/g,
'_')}.session.json`);
    this.downloadFile(schemaData, `${system.name.replace(/\s/g,
'_')}.schema.json`);
}

loadSession(event) {
    const file = event.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (e) => {
        try {
            const loadedSystem = JSON.parse(e.target.result);
            if (loadedSystem.id && loadedSystem.name &&
loadedSystem.config) {
                this.state.managedSystems[loadedSystem.id] = loadedSystem;
                this.updateSystemSelects();
                this.setActiveSystem(loadedSystem.id);
            } else { gosApiProxy.ui.showNotification('Error', 'Invalid
session file format.', 4000); }
        } catch (error) { gosApiProxy.ui.showNotification('Error', 'Error
parsing JSON file: ' + error.message, 5000); }
    };
    reader.readAsText(file);
    this.DOM.fileLoader.value = '';
}

downloadFile(content, fileName) {
    const a = window.parent.document.createElement("a");
    const file = new Blob([content], {type: 'application/json'});
    a.href = URL.createObjectURL(file);
    a.download = fileName;
    window.parent.document.body.appendChild(a);
    a.click();
    window.parent.document.body.removeChild(a);
    URL.revokeObjectURL(a.href);
}

generateSchema(system) {
    const { results, config } = system;
    const schema = {
        "$schema": "http://json-schema.org/draft-07/schema#",
        title: "Genetic Algorithm Session Schema",
        description: "Describes the structure and semantic context of a GA
workstation session file.",
        type: "object",
        properties: {

```

```
        id: { type: "string", description: "Unique identifier for the system session." },
        name: { type: "string", description: "User-defined name for the system." },
        createdAt: { type: "string", format: "date-time", description: "ISO 8601 timestamp." },
        config: {
            type: "object",
            properties: {
                d_value: { type: "integer", description: "The 'd' dimension for the 1:d^3 ratio." },
                permLength: { type: "integer" },
                populationSize: { type: "integer" },
                numGenerations: { type: "integer" },
                alphabetG: { type: "string" }
            }
        },
        results: {
            type: "object",
            properties: {
                bestPermutation: { type: "string" },
                bestFitness: { type: "number" },
                finalRatio: { type: "number" },
                targetRatio: { type: "number" },
                fitnessHistory: { type: "array" },
                runDurationMs: { type: "integer" }
            },
            statistics: results ? {
                initialFitness: results.fitnessHistory[0],
                finalFitness:
                    results.fitnessHistory[results.fitnessHistory.length - 1],
                maxImprovement: results.fitnessHistory[0] -
                    results.fitnessHistory[results.fitnessHistory.length - 1],
                averageFitness: results.fitnessHistory.reduce((a, b)
=> a + b, 0) / results.fitnessHistory.length
            } : null
        }
    };
    return schema;
}

async runEvolution() {
    this.DOM.runBtn.disabled = true; this.DOM.runBtn.textContent =
"Evolving...";
    const startTime = performance.now();
    const activeSystem =
this.state.managedSystems[this.state.activeSystemId];
    const { config } = activeSystem;

    const d = config.d_value;
    const TARGET_RATIO = 1 / (1 + Math.pow(d, 3));

    const ALPHABET_G_SET = new Set(config.alphabetG.split(''));
}
```

```
        const calculateFitness = p => Math.abs([...p].filter(c =>
ALPHABET_G_SET.has(c)).length / p.length) - TARGET_RATIO);
        const createIndividual = len => Array.from({ length: len }, () =>
this.FULL_ALPHABET[Math.floor(Math.random() *
this.FULL_ALPHABET.length)].join(''));

        let population = Array.from({ length: config.populationSize }, () =>
createIndividual(config.permLength));
        let bestOverall = { permutation: '', fitness: Infinity };
        const fitnessHistory = [];

        for (let gen = 0; gen < config.numGenerations; gen++) {
            let ratedPop = population.map(p => ({ p, fitness:
calculateFitness(p) })).sort((a, b) => a.fitness - b.fitness);
            if (ratedPop[0].fitness < bestOverall.fitness) bestOverall = {
                permutation: ratedPop[0].p, fitness: ratedPop[0].fitness };
            fitnessHistory.push(bestOverall.fitness);

            if (gen % 10 === 0 || gen === config.numGenerations - 1) {
                this.DOM.status.textContent = `Status: Evolving... Gen ${gen + 1}/${config.numGenerations}`;
                this.DOM.output.textContent = bestOverall.permutation;
                this.DOM.fitness.textContent =
bestOverall.fitness.toExponential(4);
                const finalRatio = [...bestOverall.permutation].filter(c =>
ALPHABET_G_SET.has(c)).length / config.permLength;
                this.DOM.ratio.textContent = `${finalRatio.toFixed(4)}
(Target: ${TARGET_RATIO.toFixed(4)})`;
                await new Promise(r => setTimeout(r, 0));
            }

            const newPopulation = ratedPop.slice(0,
Math.floor(config.populationSize * config.elitismRate)).map(i => i.p);
            while (newPopulation.length < config.populationSize) {
                const p1 = ratedPop[Math.floor(Math.random() * ratedPop.length
/ 2)].p;
                const p2 = ratedPop[Math.floor(Math.random() * ratedPop.length
/ 2)].p;
                const crossPoint = Math.floor(Math.random() *
config.permLength);
                let child = p1.substring(0, crossPoint) +
p2.substring(crossPoint);
                if (Math.random() < config.mutationRate) {
                    const i = Math.floor(Math.random() * config.permLength);
                    const char = this.FULL_ALPHABET[Math.floor(Math.random() *
this.FULL_ALPHABET.length)];
                    child = child.substring(0, i) + char + child.substring(i + 1);
                }
                newPopulation.push(child);
            }
            population = newPopulation;
        }
    }
```

```

        activeSystem.results = {
            bestPermutation: bestOverall.permutation,
            bestFitness: bestOverall.fitness,
            finalRatio: [...bestOverall.permutation].filter(c =>
ALPHABET_G_SET.has(c)).length / config.permLength,
            targetRatio: TARGET_RATIO,
            fitnessHistory: fitnessHistory,
            runDurationMs: Math.round(performance.now() - startTime)
        };

        this.DOM.status.textContent = `Status: Evolution complete in
${activeSystem.results.runDurationMs}ms.`;
        this.DOM.runBtn.disabled = false; this.DOM.runBtn.textContent = "▶
Run Active System";
    }
}

new GAWorkstation();
}

```

## [27] GenesisOS/digital-linguist.json

- **Bytes:** 481
- **Type:** text

```
{
  "id": "digital-linguist",
  "title": "Digital Linguist Network",
  "description": "A multi-network GA workstation to evolve permutations based on
a dynamic 1:d\u00b3 ratio. Supports creation, analysis, and composition of ratio
systems.",
  "icon": "\ud83c\udf10",
  "category": "Science",
  "version": "2.0.0",
  "entry": "digital-linguist.js",
  "permissions": [],
  "window": {
    "width": 1450,
    "height": 950,
    "resizable": true
  }
}
```

## [28] GenesisOS/docs.py

- **Bytes:** 2452
- **Type:** text

```
import os

def list_text_files():
    # Corrected missing comma between '.env' and '.gitignore'
    # Also corrected 'json' to '.json'
    extensions = [
        '.css', '.cmd', '.env', '.gitignore', '.js',
        '.php', '.hpp', '.cpp', '.md', '.py', '.txt',
        '.ps1', '.json', '.html'
    ]
    # Filter files in the current directory based on specified extensions
    return [
        f for f in os.listdir()
        if os.path.isfile(f) and any(f.endswith(ext) for ext in extensions)
    ]

def read_file_content(filename):
    try:
        with open(filename, 'r', encoding='utf-8') as file:
            return file.read()
    except Exception as e:
        return f"[Error reading {filename}]: {e}"

def sort_files(files):
    # Extract numbers from filenames and sort based on those numbers
    def extract_number(file):
        try:
            return int(os.path.splitext(file)[0])
        except ValueError:
            return float('inf') # In case of unexpected filename format, put it
at the end

    return sorted(files, key=extract_number)

def create_txt(files):
    os.makedirs("doc", exist_ok=True) # Ensure 'doc' directory exists
    output_path = os.path.join("doc", "doc.txt")

    # Prepare sections (index, filename, content)
    sections = []
    for idx, file in enumerate(files, start=1):
        content = read_file_content(file)
        sections.append((idx, file, content))

    with open(output_path, "w", encoding='utf-8') as txt_file:
        # Header
        txt_file.write("Documentation Bundle\n")
        txt_file.write("=====\\n\\n")

        # Table of Contents
        txt_file.write("Table of Contents\\n")
        txt_file.write("-----\\n")
        for idx, file, _ in sections:
```

```

        txt_file.write(f"{idx}. {file}\n")
        txt_file.write("\n\n")

    # Sections
    for idx, file, content in sections:
        header = f"[{idx}] {file}"
        txt_file.write(header + "\n")
        txt_file.write("-" * len(header) + "\n")
        txt_file.write(f";Content of {file}\n")
        txt_file.write(content)
        txt_file.write("\n\n")

if __name__ == "__main__":
    text_files = list_text_files()
    sorted_text_files = sort_files(text_files)

    create_txt(sorted_text_files) # Generate the text file

    print("Documentation updated: Created 'doc/doc.txt'")

```

## [29] GenesisOS/filesystem.php

- **Bytes:** 13151
- **Type:** text

```

<?php
/**
 * Text Reversion
 * Filesystem Module
 *
 * This module handles all API requests related to file and directory
 * operations, enforcing permissions based on the user's session.
 */

/* =====
 * HELPER FUNCTIONS
 * ===== */
/** 
 * Resolves a virtual OS path to a real server path.
 *
 * @param string $virtualPath The virtual path (e.g., /users/admin/Desktop).
 * @return string The corresponding absolute path on the server's filesystem.
 */
function resolvePath(string $virtualPath): string {
    global $config;

    // Normalize the path to remove trailing slashes and resolve relative parts.
    $virtualPath = '/' . trim($virtualPath, '/');
    $parts = array_filter(explode('/', $virtualPath), 'strlen');

```

```
$absolutes = [];

foreach ($parts as $part) {
    if ($part === '.') {
        continue;
    }
    if ($part === '..') {
        array_pop($absolutes);
    } else {
        $absolutes[] = $part;
    }
}

$normalizedPath = '/' . implode('/', $absolutes);

// Map the normalized virtual path to its real directory on the server.
if ($config['filesystem']['use_local']) {
    if (strpos($normalizedPath, '/users/') === 0) return $config['filesystem']
['user_dir'] . substr($normalizedPath, 6);
    if (strpos($normalizedPath, '/apps/') === 0) return $config['filesystem']
['apps_dir'] . substr($normalizedPath, 5);
    if (strpos($normalizedPath, '/system/') === 0) return
$config['filesystem']['system_dir'] . substr($normalizedPath, 7);
    if (strpos($normalizedPath, '/temp/') === 0) return $config['filesystem']
['temp_dir'] . substr($normalizedPath, 5);
    return $config['filesystem']['root_dir'] . $normalizedPath;
}

return $normalizedPath;
}

/***
 * Checks if the current user has permission to access a given path.
 * This is the primary security gate for all filesystem operations.
 *
 * @param string $virtualPath The virtual path being accessed.
 * @param bool $writeAccess Set to true if the operation is destructive (write,
delete).
 * @return bool True if access is granted, false otherwise.
 */
function canAccessPath(string $virtualPath, bool $writeAccess = false): bool {
    $user = getCurrentUser();
    if (!$user) return false;

    // Admins have universal access.
    if (isAdmin()) {
        return true;
    }

    // Handle write operations first.
    if ($writeAccess) {
        // Check for developer-specific write access to their own app folder.
        if (hasPermission('filesystem.write.apps.self')) {
            if (strpos($virtualPath, '/apps/' . $user['username'] . '/') === 0 ||
```

```
$virtualPath === '/apps/' . $user['username']) {
    return true;
}
}
// Check for general user write access to their home directory.
if (hasPermission('filesystem.write.user.*')) {
    if (strpos($virtualPath, '/users/' . $user['username']) === 0) {
        return true;
    }
}
// If no specific write permission matches, deny access.
return false;
}

// Handle read operations.
if (hasPermission('filesystem.read.*')) {
    return true;
}
return false;
}

/***
 * Retrieves a user's storage quota from the database.
 * @param string $username The user's username.
 * @return int The user's quota in bytes.
 */
function getUserQuota(string $username): int {
    $users = loadUserDB(); // This function is available from auth.php
    foreach ($users as $user) {
        if ($user['username'] === $username) {
            return $user['quota'] ?? 0;
        }
    }
    return 0; // Default to 0 if user not found
}

/***
 * Calculates the total disk usage for a specific user.
 * @param string $username The user's username.
 * @return int The total size of the user's files in bytes.
 */
function getUserDiskUsage(string $username): int {
    global $config;
    $userDir = $config['filesystem']['user_dir'] . '/' . $username;
    if (!is_dir($userDir)) {
        return 0;
    }

    $totalSize = 0;
    $iterator = new RecursiveIteratorIterator(new
RecursiveDirectoryIterator($userDir, FilesystemIterator::SKIP_DOTS));
    foreach ($iterator as $file) {
        if ($file->isFile()) {
```

```
        $totalSize += $file->getSize();
    }
}

return $totalSize;
}

/*
 * API HANDLER
 */
/***
 * Handles all incoming API requests for the filesystem module.
 */
function handleFileSystemAPI() {
    global $config;
    $method = $_POST['method'] ?? '';
    $params = $_POST['params'] ?? [];
    if (is_string($params)) {
        $params = json_decode($params, true) ?? [];
    }

    $path = $params['path'] ?? null;
    // ✅ FIXED: Correctly check for methods that do not require a path.
    if ($path === null && !in_array($method, ['backupUserData',
'restoreUserData'])) {
        return ['success' => false, 'message' => 'Path parameter is missing.'];
    }

    $writeAccess = in_array($method, ['writeFile', 'deleteFile',
'createDirectory', 'restoreUserData']);

    // ✅ FIXED: Correctly check permissions for path-based methods.
    if ($path !== null && !canAccessPath($path, $writeAccess)) {
        return ['success' => false, 'message' => 'Access Denied'];
    }

    $useLocalStorage = !$config['filesystem']['use_local'];

    switch ($method) {
        case 'readFile':
            if ($useLocalStorage) {
                return ['success' => true, 'data' => ['path' => $path,
'useLocalStorage' => true]];
            }
            $realPath = resolvePath($path);
            if (file_exists($realPath) && is_file($realPath)) {
                return ['success' => true, 'data' => [
                    'content' => file_get_contents($realPath),
                    'size' => filesize($realPath),
                    'modified' => filemtime($realPath)
                ]];
            }
            return ['success' => false, 'message' => 'File not found'];
    }
}
```

```
case 'writeFile':
    if ($useLocalStorage) {
        return ['success' => true, 'data' => ['path' => $path,
'useLocalStorage' => true]];
    }
    $realPath = resolvePath($path);
    $user = getCurrentUser();
    $quota = getUserQuota($user['username']);
    $usageBefore = getUserDiskUsage($user['username']);
    $existingSize = file_exists($realPath) ? filesize($realPath) : 0;
    $newSize = strlen($params['content'] ?? '');
    if ($usageBefore - $existingSize + $newSize > $quota && !isAdmin()) {
        return ['success' => false, 'message' => 'User quota exceeded'];
    }
    $dir = dirname($realPath);
    if (!is_dir($dir)) mkdir($dir, 0755, true);
    if (file_put_contents($realPath, $params['content'] ?? '') !== false)
{
    return ['success' => true, 'data' => [
        'path' => $path, 'size' => filesize($realPath), 'modified' =>
filemtime($realPath)
    ]];
}
return ['success' => false, 'message' => 'Failed to write file'];

case 'deleteFile':
    if ($useLocalStorage) {
        return ['success' => true, 'data' => ['path' => $path,
'useLocalStorage' => true]];
    }
    $realPath = resolvePath($path);
    if (file_exists($realPath)) {
        if (is_file($realPath)) {
            if (unlink($realPath)) return ['success' => true];
        } elseif (is_dir($realPath)) {
            if (count(scandir($realPath)) <= 2) {
                if (rmdir($realPath)) return ['success' => true];
            } else {
                return ['success' => false, 'message' => 'Directory is not
empty'];
            }
        }
    }
    return ['success' => false, 'message' => 'File or directory not
found'];

case 'listDirectory':
    if ($useLocalStorage) {
        return ['success' => true, 'data' => ['path' => $path,
'useLocalStorage' => true]];
    }
    $realPath = resolvePath($path);
    if (is_dir($realPath)) {
        $items = array_diff(scandir($realPath), ['.']);
    }
    return ['success' => true, 'data' => $items];
```

```
$files = []; $dirs = [];
foreach ($items as $item) {
    $itemPath = $realPath . '/' . $item;
    $itemData = ['name' => $item, 'path' => rtrim($path, '/') .
'/' . $item, 'modified' => filemtime($itemPath)];
    if (is_file($itemPath)) {
        $itemData['type'] = 'file';
        $itemData['size'] = filesize($itemPath);
        $itemData['extension'] = pathinfo($item,
PATHINFO_EXTENSION);
        $files[] = $itemData;
    } else {
        $itemData['type'] = 'directory';
        $dirs[] = $itemData;
    }
}
sort($dirs); sort($files);
return ['success' => true, 'data' => array_merge($dirs, $files)];
}
return ['success' => false, 'message' => 'Directory not found'];

case 'createDirectory':
if ($useLocalStorage) {
    return ['success' => true, 'data' => ['path' => $path,
'useLocalStorage' => true]];
}
$realPath = resolvePath($path);
if (!file_exists($realPath)) {
    if (mkdir($realPath, 0755, true)) return ['success' => true];
    return ['success' => false, 'message' => 'Failed to create
directory'];
}
return ['success' => false, 'message' => 'Directory already exists'];

case 'backupUserData':
if ($useLocalStorage) {
    return ['success' => true, 'data' => ['useLocalStorage' => true]];
}
$user = getCurrentUser();
$base = $config['filesystem']['user_dir'] . '/' . $user['username'];
$files = [];
if (file_exists($base)) {
    $iterator = new RecursiveIteratorIterator(new
RecursiveDirectoryIterator($base, FilesystemIterator::SKIP_DOTS),
RecursiveIteratorIterator::SELF_FIRST);
    foreach ($iterator as $file) {
        $rel = substr($file->getPathname(), strlen($base));
        $vpath = '/users/' . $user['username'] . $rel;
        if ($file->isDir()) {
            $files[] = ['path' => $vpath, 'type' => 'directory'];
        } else {
            $files[] = ['path' => $vpath, 'type' => 'file', 'content' => base64_encode(file_get_contents($file->getPathname()))];
        }
    }
}
```

```

        }
    }
    return [ 'success' => true, 'data' => $files];

    case 'restoreUserData':
        if ($useLocalStorage) {
            return [ 'success' => true, 'data' => [ 'useLocalStorage' => true ]];
        }
        if (isset($params['files']) && is_array($params['files'])) {
            $user = getCurrentUser();
            $base = $config['filesystem']['user_dir'] . '/' .
$user[ 'username' ];
            foreach ($params['files'] as $item) {
                $virtualPath = $item[ 'path' ] ?? '';
                if (strpos($virtualPath, '/users/' . $user[ 'username' ]) !== 0)
continue;
                $real = resolvePath($virtualPath);
                if ($item[ 'type' ] === 'directory') {
                    if (!file_exists($real)) mkdir($real, 0755, true);
                } elseif ($item[ 'type' ] === 'file') {
                    $quota = getUserQuota($user[ 'username' ]);
                    $usage = getUserDiskUsage($user[ 'username' ]);
                    $data = base64_decode($item[ 'content' ]);
                    $existing = file_exists($real) ? filesize($real) : 0;
                    if ($usage - $existing + strlen($data) > $quota &&
!isAdmin()) {
                        return [ 'success' => false, 'message' => 'User quota
exceeded' ];
                    }
                    $dir = dirname($real);
                    if (!file_exists($dir)) mkdir($dir, 0755, true);
                    file_put_contents($real, $data);
                }
            }
            return [ 'success' => true ];
        }
        return [ 'success' => false, 'message' => 'No files provided for
restore.' ];
    default:
        return [ 'success' => false, 'message' => 'Invalid filesystem method' ];
    }
}

```

### [30] GenesisOS/fsm\_pipeline.cpp

- **Bytes:** 15510
- **Type:** text

```
/**  
 * @file fsm_pipeline.cpp  
 * @brief A dynamic, range-based FSM for pipelined permutation lookups.  
 *  
 * @details  
 * This program implements a streaming Finite State Machine (FSM) that  
 * processes two input files (rule and data) to generate a sequence of outputs  
 * based on a third set of files (the word directory).  
 *  
 * It operates based on the following new principles:  
 *  
 * 1. Rule Ranging: The FSM is initialized with a specific line range  
 * (e.g., lines 1000-2000) from the <rule_file>. Only this "chunk"  
 * of '0's and '1's is used as the FSM's transition logic, looping  
 * within that chunk as needed.  
 *  
 * 2. Dynamic Lengths: The FSM no longer targets a specific digit length.  
 * Instead, it collects digits ('1', '14', '147', '1472', ...)  
 * from the <data_file> whenever the rule chunk "accepts" ('1').  
 *  
 * 3. Dynamic Lookup: After *every* accepted digit, the FSM checks its  
 * current collected digit string.  
 * - Let collected_digits = "147" (length 3).  
 * - It checks if a file exists: <words_dir>/permutations_len_3.txt  
 *  
 * 4. Fallback Logic:  
 * - If the file does NOT exist: The FSM continues. It will  
 * collect the next digit (e.g., "1472") and try again.  
 * - If the file DOES exist: The FSM attempts a lookup.  
 * - On Success: (Line 147 exists) -> The word ("foo") is used.  
 * - On Failure: (Line 147 is out of range) -> The *integer*  
 * ("147") is used as a fallback.  
 *  
 * 5. Pipelining:  
 * - When an output (word or integer) is successfully generated,  
 * it is appended to the final output string, followed by the  
 * <delimiter>.  
 * - The FSM's digit buffer is cleared, and it begins collecting  
 * a new number.  
 * - This repeats until <num_outputs_to_generate> have been found.  
 *  
 * compile: g++ -std=c++17 -o fsm_pipeline fsm_pipeline.cpp  
 *  
 * Usage:  
 * ./fsm_pipeline <rule_file> <rule_start> <rule_end> <data_file> <words_dir>  
 <output_dir> <delimiter> <num_outputs>  
 *  
 * Example:  
 * ./fsm_pipeline ./binary/permutations_len_8.txt 0 10000  
 ./numbers/permutations_len_8.txt ./words ./output "_" 10  
 *  
 * This will:  
 * 1. Load lines 0-10000 from the rule file as the FSM logic.
```

```
* 2. Read from the data file, looping as needed.  
* 3. Dynamically try to find lookups in ./words/  
* 4. Generate 10 outputs (words or fallback integers) separated by "_".  
* 5. Save the final string (e.g., "word_123_another_foo") to ./output/result.txt  
*/  
  
#include <iostream>  
#include <fstream>  
#include <string>  
#include <stdexcept>  
#include <cstdint>  
#include <sstream>  
#include <filesystem> // For std::filesystem (C++17)  
#include <cstdlib>  
#include <fstream>  
  
//=====  
// 1. UTILITY FUNCTIONS  
//=====  
  
/**  
 * @brief Creates a directory if it does not already exist.  
 * @param path The path of the directory to create.  
 */  
static void create_directory_if_not_exists(const std::string& path) {  
    std::error_code ec;  
    if (!std::filesystem::create_directory(path, ec) && ec) {  
        if (ec != std::errc::file_exists) {  
            throw std::runtime_error("Failed to create directory: " + path + " - "  
+ ec.message());  
        }  
    }  
}  
  
/**  
 * @brief Reads a specific line from a file (0-based index).  
 * @param filepath The path to the file.  
 * @param line_index The 0-based line index to retrieve.  
 * @param[out] out_line String to store the result.  
 * @return true on success, false if line_index is out of range.  
 * @throws std::runtime_error if the file cannot be opened.  
 */  
static bool read_line_from_file(const std::string& filepath, std::uint64_t  
line_index, std::string& out_line) {  
    std::ifstream file(filepath);  
    if (!file.is_open()) {  
        throw std::runtime_error("Failed to open lookup file: " + filepath);  
    }  
  
    std::uint64_t current_index = 0;  
    while (std::getline(file, out_line)) {  
        if (current_index == line_index) {  
            file.close();  
            return true;  
        }  
        current_index++;  
    }  
}
```

```
        }
        current_index++;
    }

    file.close();
    return false; // Line index was out of range
}

//=====
// 2. CORE LOGIC (PermutationFSM)
//=====

/***
 * @class PermutationFSM
 * @brief Implements the FSM logic for dynamic, pipelined lookups.
 */
class PermutationFSM {
private:
    // Input Streams
    std::ifstream m_data_stream;
    std::string m_data_filepath;
    std::string m_rule_chunk; // Holds the specified range of the rule file
    std::uint64_t m_rule_index = 0; // Cursor for m_rule_chunk

    // Configuration
    std::string m_words_dir;
    std::string m_delimiter;
    std::uint64_t m_num_outputs;

    /**
     * @brief Loads the specified line range from the rule file into memory.
     * @note This concatenates all '0's and '1's from the lines,
     *       ignoring newlines within the range.
     */
    void load_rule_chunk(const std::string& rule_file, std::uint64_t start,
std::uint64_t end) {
        std::ifstream file(rule_file);
        if (!file.is_open()) {
            throw std::runtime_error("Failed to open rule file: " + rule_file);
        }

        std::string line;
        std::uint64_t current_line = 0;
        std::stringstream ss;

        while (std::getline(file, line)) {
            if (current_line > end) {
                break; // We are past the specified range
            }
            if (current_line >= start) {
                ss << line; // Append the line's content
            }
            current_line++;
        }
    }
}
```

```
file.close();

    m_rule_chunk = ss.str();
    if (m_rule_chunk.empty()) {
        throw std::runtime_error("Rule chunk is empty. Check rule file and
line range.");
    }
}

/***
 * @brief Gets the next '0' or '1' from the in-memory rule chunk.
 * @note Loops back to the start of the chunk on (chunk) EOF.
 */
char get_next_rule_char() {
    if (m_rule_index >= m_rule_chunk.length()) {
        m_rule_index = 0; // Loop back to start
    }
    return m_rule_chunk[m_rule_index++];
}

/***
 * @brief Gets the next valid digit from the data stream.
 * @note Skips newlines and loops back to the start on (file) EOF.
 */
char get_next_data_char() {
    char c;
    while (true) {
        if (m_data_stream.get(c)) {
            if (c == '\n' || c == '\r') {
                continue; // Skip newlines
            }
            return c; // Got a valid character
        } else {
            // Reached End-of-File
            m_data_stream.clear(); // Clear EOF flags
            m_data_stream.seekg(0); // Rewind to the beginning of the file
        }
    }
}

/***
 * @brief Checks if a lookup file exists for the given length and,
 * if so, attempts to find the word.
 * @param digits The collected digit string (e.g., "147").
 * @return The found word, the fallback integer string, or an
 * empty string (if no file exists yet).
 */
std::string attempt_lookup(const std::string& digits) {
    std::uint64_t len = digits.length();
    if (len == 0) {
        return ""; // Should not happen, but safe to check
    }

    // 1. Check if the required word file exists
```

```
    std::string word_filepath = m_words_dir + "/permutations_len_" +
std::to_string(len) + ".txt";

    if (!std::filesystem::exists(word_filepath)) {
        return ""; // File does not exist for this length. Keep collecting.
    }

    // 2. File exists. Attempt the lookup.
    try {
        std::uint64_t line_index = std::stoull(digits);
        std::string result_word;

        if (read_line_from_file(word_filepath, line_index, result_word)) {
            // Success: Word was found
            return result_word;
        } else {
            // Failure: Line index out of range. Use fallback.
            return digits;
        }
    }
    catch (const std::exception& e) {
        // Failure: e.g., stoull overflow ("999...999"). Use fallback.
        return digits;
    }
}

public:
PermutationFSM(
    const std::string& rule_file,
    std::uint64_t rule_start,
    std::uint64_t rule_end,
    const std::string& data_file,
    const std::string& words_dir,
    const std::string& delimiter,
    std::uint64_t num_outputs)
: m_data_filepath(data_file),
m_words_dir(words_dir),
m_delimiter(delimiter),
m_num_outputs(num_outputs) {

    // 1. Load the rule chunk
    load_rule_chunk(rule_file, rule_start, rule_end);

    // 2. Open the data stream
    m_data_stream.open(m_data_filepath);
    if (!m_data_stream.is_open()) {
        throw std::runtime_error("Failed to open data file: " +
m_data_filepath);
    }
}

~PermutationFSM() {
    if (m_data_stream.is_open()) {
        m_data_stream.close();
```

```
        }

    }

    /**
     * @brief Runs the main FSM pipeline.
     * @return The final, concatenated output string.
     */
    std::string run_pipeline() {
        std::stringstream final_output_ss;
        std::string collected_digits;
        std::uint64_t outputs_generated = 0;

        while (outputs_generated < m_num_outputs) {
            // State: Get inputs
            char rule_c = get_next_rule_char();
            char data_c = get_next_data_char();

            // FSM Logic
            if (rule_c == '1') {
                // State: Accept
                collected_digits.push_back(data_c);

                // Action: Try to resolve the new digit string
                std::string result_item = attempt_lookup(collected_digits);

                if (!result_item.empty()) {
                    // A lookup was successful (found word or fallback int)

                    if (outputs_generated > 0) {
                        final_output_ss << m_delimiter;
                    }
                    final_output_ss << result_item;

                    outputs_generated++;
                    collected_digits.clear(); // Reset for next item
                }
                // else (result_item is empty)
                // File for this length didn't exist.
                // Do nothing; loop will continue collecting.
            }
            // else (rule_c == '0')
            // State: Reject
            // Action: Do nothing.
        }

        return final_output_ss.str();
    }

    // Disable copy and assignment
    PermutationFSM(const PermutationFSM&) = delete;
    PermutationFSM& operator=(const PermutationFSM&) = delete;
};

//=====
```

```
// 3. SYSTEM ASSEMBLER (main)
//=====

static void print_usage(const char* program_name) {
    std::cerr << "Usage: " << program_name
        << " <rule_file> <rule_start_line> <rule_end_line> <data_file>"
        << " <words_dir> <output_dir> <delimiter>
<num_outputs_to_generate>\n\n"
        << "Example: " << program_name
        << " ./binary/permutations_len_8.txt 0 10000"
        << " ./numbers/permutations_len_8.txt"
        << " ./words ./output \"_\" 10\n\n"
        << "This generates 10 outputs (words or integers) separated by
\"_\".\n";
}

int main(int argc, char* argv[]) {
    // --- DEBUG: basic lifecycle logging ---
    {
        std::ofstream dbg("fsm_debug.txt", std::ios::app);
        dbg << "fsm_pipeline started. argc = " << argc << "\n";
    }

    // --- 1. Argument Validation ---
    if (argc != 9) {
        {
            std::ofstream dbg("fsm_debug.txt", std::ios::app);
            dbg << "argc != 9, calling print_usage and exiting.\n";
        }
        print_usage(argv[0]);
        return EXIT_FAILURE;
    }

    // --- 2. Parse Runtime Parameters ---
    try {
        std::string rule_file = argv[1];
        std::uint64_t rule_start = std::stoull(argv[2]);
        std::uint64_t rule_end   = std::stoull(argv[3]);
        std::string data_file   = argv[4];
        std::string words_dir   = argv[5];
        std::string output_dir  = argv[6];
        std::string delimiter   = argv[7];
        std::uint64_t num_outputs = std::stoull(argv[8]);

        {
            std::ofstream dbg("fsm_debug.txt", std::ios::app);
            dbg << "Parsed args OK. rule_file = " << rule_file
                << ", data_file = " << data_file
                << ", num_outputs = " << num_outputs << "\n";
        }
    }

    // --- 3. Safety Checks ---
    if (rule_end < rule_start) {
        throw std::invalid_argument("rule_end_line must be >=

```

```
rule_start_line.");
    }
    if (num_outputs == 0) {
        throw std::invalid_argument("num_outputs_to_generate must be > 0.");
    }

    // --- 4. System Assembly & Execution ---
    std::cout << "Initializing FSM Pipeline...\n"
        << "    Rule File: " << rule_file << "\n"
        << "    Rule Range: " << rule_start << " to " << rule_end << "\n"
        << "    Data File: " << data_file << "\n"
        << "    Words Dir: " << words_dir << "\n"
        << "    Output Dir: " << output_dir << "\n"
        << "    Delimiter: \" " << delimiter << "\"\n"
        << "    Target Outputs: " << num_outputs << "\n\n";

    // a. Instantiate FSM (loads rule chunk, opens data file)
    PermutationFSM fsm(
        rule_file, rule_start, rule_end,
        data_file, words_dir, delimiter, num_outputs
    );

    // b. Run the pipeline
    std::cout << "Running pipeline to generate " << num_outputs << " outputs..." << std::flush;
    std::string final_output_string = fsm.run_pipeline();
    std::cout << " Done.\n";

    // c. Write the result to the output file
    create_directory_if_not_exists(output_dir);
    std::string output_filepath = output_dir + "/result.txt";

    std::ofstream out_stream(output_filepath, std::ios::out | std::ios::trunc);
    if (!out_stream.is_open()) {
        throw std::runtime_error("Failed to open output file: " +
output_filepath);
    }
    out_stream << final_output_string << std::endl;
    out_stream.close();

    std::cout << "\nResult successfully saved to: " << output_filepath << "\n"
        << "Output String: " << final_output_string.substr(0, 200) <<
"...\n";

} catch (const std::exception& e) {
{
    std::ofstream dbg("fsm_debug.txt", std::ios::app);
    dbg << "Exception caught: " << e.what() << "\n";
}

std::cerr << "\n\n--- A fatal error occurred ---\n"
        << "Error: " << e.what() << "\n";
print_usage(argv[0]);
}
```

```
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

### [31] [GenesisOS/improvements\\_to\\_be\\_made.md](#)

- **Bytes:** 7330
- **Type:** text

Based on the provided files, the **Genesis OS (Text Reversion)** codebase is a **web-based operating system simulation**. It attempts to recreate the desktop computing paradigm—complete with a window manager, file system, process isolation, and applications—within a browser environment, backed by a PHP monolithic backend.

Below is the detailed working philosophy and a roadmap for upgrading this system for an enterprise production release.

---

#### [### I. Working Philosophy](#)

The architectural philosophy of this codebase relies on **Virtualization** and **stateful abstractions** to mimic a desktop OS over a stateless web protocol (HTTP).

##### [#### 1. The "Browser-as-Kernel" Metaphor](#)

The frontend (`tr-script.js`, `system.php`) acts as the "kernel." It manages:

- \* **Process Isolation:** Applications are loaded into ``<iframe>`` elements to sandbox their DOM and CSS execution from the main system .

- \* **Virtual Hardware:** It mocks hardware interfaces (like `stdout` in `tr-terminal.js`) and window management (minimize, maximize, focus) via the DOM.

- \* **System Calls:** Applications do not fetch data directly; they communicate with the "kernel" (the main window) via a global proxy (`gosApiProxy`), which then routes requests to the PHP backend .

##### [#### 2. The Virtual Filesystem \(VFS\)](#)

The backend abstracts the server's local file system into a virtual user-space file system.

- \*  
**Path Resolution:** It maps virtual paths (e.g., `/users/admin/Desktop`) to physical server paths using `resolvePath()` .
  
- \*  
**Jail/Chroot:** Security relies on "jailing" users to their directories (`gos\_files/users/<username>`) and preventing directory traversal attacks programmatically rather than at the OS level .

#### #### 3. Flat-File Data Persistence

The system avoids complex database dependencies by using **JSON flat files** for critical data:

- \*  
**Identity:** Users and roles are stored in `users.json` .
  
- \*  
**Registry:** Application manifests and submissions are stored in `app\_submissions.json` .
  
- \*  
**State:** Configuration is hardcoded in PHP arrays (`config.php`) or JSON.

#### #### 4. Modular Monolith Backend

The backend is structured as a modular monolith. A central entry point (`index.php` or `tr-api.php`) routes requests to specific modules (`auth.php`, `filesystem.php`, `apps.php`) based on a `method` parameter. There is no rigid MVC structure; functions are essentially RPC endpoints.

---

### ## II. Improvements for Enterprise Production Release

To move from a "simulation" prototype to a secure, scalable enterprise platform, the following transformations are required:

#### #### 1. Data Layer: From JSON to Relational Database

- Current State:** User data and app registries are stored in JSON files.
- Risk:** JSON files lack ACID compliance, handle concurrent writes poorly (file locking issues), and scale poorly as user counts grow.
- Enterprise Fix:**

- \* **Migration:** Move `users.json` and `app\_submissions.json` to a relational database (PostgreSQL or MySQL).

\* **ORM:** Implement an Object-Relational Mapper (like Eloquent or Doctrine) to manage data access, replacing raw `json\_decode/encode` calls.

#### #### 2. Security Hardening: The "Sandbox" Problem

**Current State:** The system executes user-submitted PHP code using `proc\_open` and `tempnam` files on the host server. It attempts to secure this via `disable\_functions`.

**Risk:** This is a critical vulnerability. `disable\_functions` is frequently bypassed. Executing untrusted code on the host filesystem is dangerous in a production environment.

**Enterprise Fix:**

\* **Containerization:** Move the execution environment (Sandbox) off the host. Use **Docker containers** or **MicroVMs (Firecracker)** for executing user scripts.

The PHP backend should queue a job, and an isolated worker should execute the code and return the result.

\* **Read-Only Filesystems:** The production container should have a read-only filesystem, with writable areas limited to temporary RAM disks.

#### #### 3. Filesystem Abstraction: From Local Disk to Object Storage

**Current State:** Files are stored directly on the server's disk under `gos\_files/`.

**Risk:** This prevents horizontal scaling. If you add a second load-balanced server, it won't have access to files uploaded to the first server.

**Enterprise Fix:**

\* **Abstraction Layer:** Implement the **Flysystem** PHP library.

\* **Object Storage:** Configure the filesystem module to write to an S3-compatible service (AWS S3, MinIO) instead of local disk. This allows the application to be stateless and scalable.

#### #### 4. Dependency Management & Standards

**Current State:** The codebase relies on manual `require\_once` calls and appears to have zero external dependencies managed by a package manager.

**Risk:** Reinventing the wheel (e.g., custom routing, custom logging) leads to maintenance debt and security holes.

**Enterprise Fix:**

\* **Composer:** Introduce `composer.json`. Use established packages for logging (Monolog), HTTP clients (Guzzle), and input validation.

\* **PSR Standards:** Refactor the code to follow PSR-4 (Autoloading) and PSR-12 (Coding Style) to make it maintainable by other PHP developers.

#### #### 5. Identity & Access Management (IAM)

**Current State:** Custom session handling with a JSON user database.

**Risk:** Storing password hashes in a flat file and managing sessions manually is prone to implementation errors.

**Enterprise Fix:**

\* **SSO/OIDC:** Integrate with an enterprise identity provider (Auth0, Okta, or

Active Directory) using OpenID Connect.

\* **Session Store:** Move PHP sessions to **Redis** to allow session persistence across server restarts and load balancers.

#### #### 6. CI/CD & Version Control

**Current State:** The system includes a custom `VersionManager` class that copies directories to create "snapshots".

**Risk:** This reinvents Git poorly. It wastes disk space and lacks branching/merging capabilities.

**Enterprise Fix:**

\* **Git-based Workflow:** Remove `tr-VersionManager.php`. Use Git for version control.

\* **Deployment Pipeline:** Implement a CI/CD pipeline (GitHub Actions/GitLab CI) that runs tests (PHPUnit), builds assets, and deploys artifacts to the server.

#### #### 7. Frontend Architecture

**Current State:** The frontend injects raw HTML strings into the DOM and manages state via global variables or DOM queries.

**Enterprise Fix:**

\* **Component Framework:** Refactor the frontend into a component-based framework (React, Vue, or Svelte). This will manage the "window" state and DOM updates much more efficiently than raw string injection.

\* **Build System:** Use Vite or Webpack to bundle assets, enabling TypeScript support for better type safety in the "kernel" logic.

#### ### Summary of Transformation

Feature	Current Philosophy	Enterprise Philosophy
---	---	---
<b>Storage</b>	Local JSON & Filesystem	SQL Database & S3 Object Storage
<b>Code Exec</b>	`proc_open` on Host	Docker/Firecracker MicroVMs
<b>Scaling</b>	Vertical (Single Server)	Horizontal (Stateless App + Redis + CDN)
<b>Auth</b>	Custom JSON DB	OAuth2 / OIDC / LDAP
<b>Codebase</b>	Custom Monolith	Composer-managed, PSR-compliant, Framework-based (e.g., Symfony/Laravel)

[32] [GenesisOS/index.css](#)

- **Bytes:** 22952
- **Type:** text

```
/**
 * Text Reversion
 * Main Stylesheet
 *
 * Build: 20250711
```

```
/*
 * =====
 * BASE & THEME SETUP
 * ===== */
/* Reset and Base Styles */
*, *::before, *::after {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

html,
body {
    height: 100%;
    width: 100%;
    overflow: hidden;
    font-family: 'Segoe UI', 'Arial', sans-serif;
    font-size: 16px;
    line-height: 1.5;
}

/* Theme Variables */
:root {
    /* Vintage Theme (Default) */
    --main-bg: #000000;
    --main-text: #33ff33;
    --main-border: #33ff33;
    --terminal-bg: #111111;
    --button-bg: #333333;
    --highlight: #00aa00;
    --cell-default-bg: #222222;
    --cell-default-text: #33ff33;
    --window-header: #005500;
    --window-body: rgba(0, 17, 0, 0.9);
    --icon-bg: #001100;
    --tooltip-bg: #001100;
    --tooltip-border: #00aa00;
    --scrollbar-thumb: #00aa00;
    --scrollbar-track: #001100;
    --menu-bg: rgba(0, 17, 0, 0.95);
}

/* Dark theme class */
.theme-dark {
    --main-bg: #1e1e1e;
    --main-text: #f0f0f0;
    --main-border: #444444;
    --terminal-bg: #252525;
    --button-bg: #333333;
    --highlight: #3f51b5;
```

```
--cell-default-bg: #333333;
--cell-default-text: #f0f0f0;
--window-header: #252525;
--window-body: rgba(30, 30, 30, 0.9);
--icon-bg: #252525;
--tooltip-bg: #252525;
--tooltip-border: #3f51b5;
--scrollbar-thumb: #3f51b5;
--scrollbar-track: #252525;
--menu-bg: rgba(30, 30, 30, 0.95);
}

/* Light theme class */
.theme-light {
    --main-bg: #f0f0f0;
    --main-text: #333333;
    --main-border: #cccccc;
    --terminal-bg: #ffffff;
    --button-bg: #e0e0e0;
    --highlight: #3f51b5;
    --cell-default-bg: #ffffff;
    --cell-default-text: #333333;
    --window-header: #e0e0e0;
    --window-body: rgba(255, 255, 255, 0.9);
    --icon-bg: #e0e0e0;
    --tooltip-bg: #ffffff;
    --tooltip-border: #3f51b5;
    --scrollbar-thumb: #3f51b5;
    --scrollbar-track: #e0e0e0;
    --menu-bg: rgba(255, 255, 255, 0.95);
}

/* Blue theme class */
.theme-blue {
    --main-bg: #1a2130;
    --main-text: #33aaff;
    --main-border: #33aaff;
    --terminal-bg: #0c1020;
    --button-bg: #253040;
    --highlight: #0077cc;
    --cell-default-bg: #1c2636;
    --cell-default-text: #33aaff;
    --window-header: #003366;
    --window-body: rgba(10, 20, 40, 0.9);
    --icon-bg: #0c1020;
    --tooltip-bg: #0c1020;
    --tooltip-border: #0077cc;
    --scrollbar-thumb: #0077cc;
    --scrollbar-track: #0c1020;
    --menu-bg: rgba(10, 20, 40, 0.95);
}

body {
    background-color: var(--main-bg);
```

```
        color: var(--main-text);  
    }  
  
    /* Scrollbar styling */  
    ::-webkit-scrollbar {  
        width: 10px;  
        height: 10px;  
    }  
    ::-webkit-scrollbar-track {  
        background: var(--scrollbar-track);  
    }  
    ::-webkit-scrollbar-thumb {  
        background: var(--scrollbar-thumb);  
        border-radius: 2px;  
    }  
    ::-webkit-scrollbar-thumb:hover {  
        background: var(--highlight);  
    }  
  
/* ======  
 * SYSTEM SCREENS (SPLASH, LOGIN, ERROR)  
 * ====== */  
  
/* Splash Screen */  
.splash-screen {  
    position: fixed;  
    top: 0;  
    left: 0;  
    width: 100%;  
    height: 100%;  
    background-color: var(--main-bg);  
    display: flex;  
    flex-direction: column;  
    justify-content: center;  
    align-items: center;  
    z-index: 9999;  
    transition: opacity 0.5s ease-in-out;  
}  
.splash-screen.fade-out {  
    opacity: 0;  
}  
#splash-logo {  
    font-size: 80px;  
    margin-bottom: 40px;  
    text-shadow: 0 0 15px var(--main-text);  
    animation: pulsate 2s infinite;  
}  
@keyframes pulsate {  
    0% { opacity: 0.8; }  
    50% { opacity: 1.0; text-shadow: 0 0 20px var(--main-text); }  
    100% { opacity: 0.8; }  
}  
.loading-indicator {  
    width: 300px;
```

```
    text-align: center;
}
.loading-bar {
  width: 100%;
  height: 6px;
  background-color: var(--terminal-bg);
  border-radius: 3px;
  overflow: hidden;
  margin-bottom: 10px;
}
.loading-progress {
  height: 100%;
  width: 0;
  background: var(--highlight);
  border-radius: 3px;
  transition: width 0.3s ease-out;
}
.loading-status {
  font-size: 14px;
}

/* Login Screen */
.login-screen {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: var(--main-bg);
  background-image: radial-gradient(circle at center, var(--button-bg) 0%, var(--main-bg) 100%);
  z-index: 9998;
  opacity: 0;
  transition: opacity 0.5s ease-in-out;
}
.login-screen.show {
  display: flex;
  justify-content: center;
  align-items: center;
  opacity: 1;
}
.login-container {
  width: 360px;
  padding: 30px;
  background-color: var(--window-body);
  border-radius: 10px;
  border: 1px solid var(--main-border);
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.5);
}
.login-logo {
  text-align: center;
  margin-bottom: 30px;
  font-size: 50px;
```

```
}

.login-heading {
    text-align: center;
    font-size: 24px;
    font-weight: 400;
    margin-bottom: 30px;
}

.login-form {
    display: flex;
    flex-direction: column;
}

.login-input {
    position: relative;
    margin-bottom: 20px;
}

.login-input input {
    width: 100%;
    padding: 10px 15px;
    font-size: 16px;
    background-color: var(--terminal-bg);
    color: var(--main-text);
    border: 1px solid var(--main-border);
    border-radius: 4px;
    outline: none;
}

.login-input input:focus {
    box-shadow: 0 0 0 2px var(--highlight);
}

.login-button {
    padding: 12px;
    background-color: var(--highlight);
    color: var(--main-bg);
    font-size: 16px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: opacity 0.3s;
}

.login-button:hover {
    opacity: 0.9;
}

.login-error {
    color: #ff4081;
    font-size: 14px;
    text-align: center;
    margin-top: 15px;
    min-height: 20px;
}

/* Error Screen */
.error-screen {
    position: fixed;
    top: 0;
    left: 0;
```

```
width: 100%;  
height: 100%;  
background-color: var(--main-bg);  
display: none;  
justify-content: center;  
align-items: center;  
z-index: 9999;  
}  
.error-container {  
width: 500px;  
padding: 30px;  
background-color: var(--window-body);  
border: 1px solid var(--main-border);  
border-radius: 10px;  
box-shadow: 0 10px 30px rgba(0, 0, 0, 0.5);  
text-align: center;  
}  
.error-container h2 {  
color: #f44336;  
margin-bottom: 20px;  
}  
.error-container p {  
margin-bottom: 25px;  
opacity: 0.8;  
}  
.error-container pre {  
text-align: left;  
background-color: var(--terminal-bg);  
padding: 15px;  
border-radius: 4px;  
overflow: auto;  
margin-bottom: 25px;  
white-space: pre-wrap;  
word-break: break-all;  
}  
#restart-button {  
padding: 10px 20px;  
background-color: var(--highlight);  
color: var(--main-bg);  
border: none;  

```

```
top: 0;
left: 0;
width: 100%;
height: 100%;
background-color: var(--main-bg);
background-image:
    radial-gradient(var(--main-border) 1px, transparent 1px),
    radial-gradient(var(--main-border) 1px, transparent 1px);
background-size: 50px 50px;
background-position: 0 0, 25px 25px;
background-attachment: fixed;
opacity: 0.8;
overflow: hidden;
display: none;
opacity: 0;
transition: opacity 0.5s ease-in-out;
}

.desktop.show {
    display: block;
    opacity: 1;
}

/* Desktop Icons */
.desktop-icons {
    display: grid;
    grid-template-columns: repeat(auto-fill, 80px);
    grid-gap: 20px;
    padding: 20px;
    position: absolute;
    top: 0;
    left: 0;
}
.desktop-icon {
    width: 80px;
    display: flex;
    flex-direction: column;
    align-items: center;
    cursor: pointer;
    padding: 5px;
    border-radius: 5px;
}
.desktop-icon:hover {
    background-color: rgba(var(--highlight), 0.3);
}
.icon-image {
    width: 48px;
    height: 48px;
    background-color: var(--icon-bg);
    border: 1px solid var(--main-border);
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 24px;
    margin-bottom: 5px;
}
```

```
    color: var(--main-text);
}

.icon-text {
    text-align: center;
    font-size: 12px;
    word-wrap: break-word;
    max-width: 80px;
    text-shadow: 1px 1px 2px black;
}

/* Workspace */

.workspace {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 40px; /* Space for taskbar */
    overflow: hidden;
}

/* Taskbar */

.taskbar {
    position: absolute;
    bottom: 0;
    left: 0;
    right: 0;
    height: 40px;
    background-color: var(--terminal-bg);
    border-top: 1px solid var(--main-border);
    display: flex;
    z-index: 1000;
}
.start-button {
    width: 60px;
    height: 40px;
    display: flex;
    justify-content: center;
    align-items: center;
    background: none;
    border: none;
    color: var(--main-text);
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.2s;
}
.start-button:hover {
    background-color: var(--highlight);
    color: var(--main-bg);
}
.taskbar-items {
    flex: 1;
    display: flex;
    padding: 0 5px;
    overflow-x: auto;
```

```
    overflow-y: hidden;
}
.taskbar-item {
    height: 40px;
    min-width: 160px;
    max-width: 200px;
    display: flex;
    align-items: center;
    padding: 0 10px;
    margin-right: 4px;
    background-color: rgba(60, 60, 60, 0.3);
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.2s;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}
.taskbar-item:hover {
    background-color: rgba(80, 80, 80, 0.5);
}
.taskbar-item.active {
    background-color: var(--highlight);
    color: var(--main-bg);
}
.taskbar-item-icon {
    margin-right: 8px;
    font-size: 16px;
}
.taskbar-item-title {
    flex: 1;
    overflow: hidden;
    text-overflow: ellipsis;
}
.system-tray {
    display: flex;
    align-items: center;
    padding: 0 10px;
}
.clock {
    font-size: 14px;
    min-width: 55px;
    text-align: center;
}
/* Start Menu */
.start-menu {
    position: absolute;
    bottom: 40px;
    left: 0;
    width: 300px;
    max-height: 0;
    background-color: var(--menu-bg);
    border-radius: 0 6px 0 0;
```

```
border: 1px solid var(--main-border);
border-bottom: none;
box-shadow: 0 0 20px rgba(0, 0, 0, 0.5);
overflow: hidden;
z-index: 1001;
transition: max-height 0.3s ease-out;
display: flex; /* Add this */
flex-direction: column; /* Add this */
}
.start-menu.active {
  max-height: 600px;
}
.user-info {
  padding: 20px;
  display: flex;
  align-items: center;
  background: linear-gradient(135deg, var(--highlight) 0%, var(--main-border) 100%);
  color: var(--main-bg);
}
.user-avatar {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  background-color: var(--main-bg);
  margin-right: 15px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 20px;
}
.user-name {
  font-size: 16px;
  font-weight: 500;
}
}
.menu-items {
  padding: 10px 0;
  overflow-y: auto; /* Make the list scrollable */
  max-height: 400px; /* Set a max height before scrolling starts */
}
.menu-item {
  padding: 10px 20px;
  cursor: pointer;
  transition: background-color 0.2s;
  display: flex;
  align-items: center;
}
.menu-item:hover {
  background-color: var(--highlight);
  color: var(--main-bg);
}
.menu-item-icon {
  margin-right: 10px;
  font-size: 16px;
}
```

```
width: 20px;
text-align: center;
}
.menu-separator {
height: 1px;
background-color: var(--main-border);
margin: 10px 0;
opacity: 0.3;
}

/* =====
* WINDOW SYSTEM
* ===== */

.window {
position: absolute;
background-color: var(--window-body);
border: 1px solid var(--main-border);
border-radius: 6px;
box-shadow: 0 5px 25px rgba(0, 0, 0, 0.4);
display: flex;
flex-direction: column;
min-width: 300px;
min-height: 200px;
z-index: 100;
transition: transform 0.3s, opacity 0.3s;
}
.window.minimized {
transform: scale(0.7);
opacity: 0;
pointer-events: none;
}
.window.maximized {
top: 0 !important;
left: 0 !important;
width: 100% !important;
height: calc(100% - 40px) !important;
border-radius: 0;
}
.window-header {
height: 32px;
background-color: var(--window-header);
border-radius: 6px 6px 0 0;
display: flex;
align-items: center;
padding: 0 10px;
cursor: move;
}
.window-title {
flex: 1;
font-size: 14px;
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
```

```
    user-select: none;
}

.window-controls {
    display: flex;
}

.window-control {
    width: 24px;
    height: 24px;
    background: none;
    border: none;
    color: var(--main-text);
    font-size: 14px;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 4px;
    margin-left: 2px;
}

.window-control:hover {
    background-color: var(--highlight);
    color: var(--main-bg);
}

.window-content {
    flex: 1;
    overflow: auto;
    position: relative;
}

.app-loading {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    background-color: var(--window-body);
    color: var(--main-text);
    font-size: 14px;
}

/* =====
 * UI COMPONENTS
 * ===== */

/* Context Menu */
.context-menu {
    position: absolute;
    min-width: 150px;
    background-color: var(--menu-bg);
    border: 1px solid var(--main-border);
    border-radius: 4px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
```

```
z-index: 2000;
padding: 5px 0;
}
.context-menu-item {
  padding: 8px 15px;
  cursor: pointer;
  transition: background-color 0.2s;
}
.context-menu-item:hover {
  background-color: var(--highlight);
  color: var(--main-bg);
}
.context-menu-separator {
  height: 1px;
  background-color: var(--main-border);
  margin: 5px 0;
  opacity: 0.3;
}

/* Notifications & Banners */
.notification-container {
  position: fixed;
  bottom: 50px;
  right: 10px;
  width: 300px;
  z-index: 1500;
  display: flex;
  flex-direction: column;
  gap: 10px;
}
.error-banner {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  background-color: #f44336;
  color: #ffffff;
  padding: 10px;
  text-align: center;
  z-index: 2000;
  display: none;
}
.debug-console {
  position: fixed;
  bottom: 0;
  left: 0;
  right: 0;
  height: 200px;
  background: #111;
  color: #0f0;
  font-family: monospace;
  padding: 5px;
  overflow: auto;
  z-index: 2500;
```

```
    display: none;
}

.notification {
  background-color: var(--window-body);
  border: 1px solid var(--main-border);
  border-radius: 6px;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
  padding: 15px;
  transform: translateX(120%);
  transition: transform 0.3s ease-out;
}

.notification.show {
  transform: translateX(0);
}

.notification-title {
  font-weight: 500;
  margin-bottom: 5px;
}

.notification-body {
  font-size: 14px;
  opacity: 0.8;
}

/* Modal Dialog */
.modal-backdrop {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.6);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 10000; /* ☑ FIXED: Increased z-index to appear above login screen */
}
  opacity: 0;
  pointer-events: none;
  transition: opacity 0.3s;
}

.modal-backdrop.show {
  opacity: 1;
  pointer-events: auto;
}

.modal {
  width: 400px;
  background-color: var(--window-body);
  border: 1px solid var(--main-border);
  border-radius: 6px;
  box-shadow: 0 5px 25px rgba(0, 0, 0, 0.5);
  transform: translateY(-20px);
  transition: transform 0.3s;
}

.modal-backdrop.show .modal {
```

```
    transform: translateY(0);
}

.modal-header {
  padding: 15px;
  border-bottom: 1px solid var(--main-border);
}

.modal-title {
  font-size: 18px;
  font-weight: 500;
}

.modal-body {
  padding: 15px;
  max-height: 70vh;
  overflow-y: auto;
}

.modal-footer {
  padding: 15px;
  border-top: 1px solid var(--main-border);
  display: flex;
  justify-content: flex-end;
  gap: 10px;
}

.button {
  padding: 8px 15px;
  border: 1px solid var(--main-border);
  border-radius: 4px;
  cursor: pointer;
  font-size: 14px;
  transition: background-color 0.2s;
  background-color: var(--button-bg);
  color: var(--main-text);
}

.button-primary {
  background-color: var(--highlight);
  color: var(--main-bg);
  border-color: var(--highlight);
}

.button:hover {
  opacity: 0.9;
}

/* =====
 * APPLICATION-SPECIFIC STYLES
 * ===== */

/* Mathematical Sandbox */

.grid-container {
  border: 1px solid var(--main-border);
  overflow: auto;
  flex: 1;
}

.grid {
  display: grid;
  grid-template-columns: repeat(1, 1fr);
```

```
grid-gap: 1px;
background-color: var(--main-border);
padding: 1px;
min-width: 100%;

}

.cell {
    background-color: var(--cell-default-bg);
    color: var(--cell-default-text);
    min-height: 50px;
    display: flex;
    align-items: center;
    justify-content: center;
    text-align: center;
    padding: 5px;
    position: relative;
    cursor: pointer;
    overflow: hidden;
    font-size: 12px;
}

.cell.split {
    padding: 0;
    display: grid;
    grid-template-columns: repeat(1, 1fr);
    grid-gap: 1px;
    background-color: var(--main-border);
}

.sub-cell {
    background-color: var(--cell-default-bg);
    display: flex;
    align-items: center;
    justify-content: center;
    min-height: 20px;
    text-align: center;
    cursor: pointer;
    padding: 2px;
}

/* File Manager */

.file-browser {
    display: flex;
    flex-direction: column;
    height: 100%;
}

.file-toolbar {
    display: flex;
    align-items: center;
    height: 40px;
    padding: 0 10px;
    background-color: var(--terminal-bg);
    border-bottom: 1px solid var(--main-border);
}

.file-browser-content {
    display: flex;
    flex: 1;
```

```
}

.file-sidebar {
  width: 200px;
  background-color: var(--terminal-bg);
  border-right: 1px solid var(--main-border);
  overflow-y: auto;
}

.file-sidebar-item {
  padding: 8px 15px;
  cursor: pointer;
  display: flex;
  align-items: center;
}

.file-sidebar-item:hover {
  background-color: var(--highlight);
  color: var(--main-bg);
}

.file-sidebar-item.active {
  background-color: var(--highlight);
  color: var(--main-bg);
}

.file-sidebar-icon {
  margin-right: 10px;
  font-size: 16px;
}

.file-main {
  flex: 1;
  overflow: auto;
  padding: 10px;
}

.file-list {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
  gap: 10px;
}

.file-item {
  display: flex;
  flex-direction: column;
  align-items: center;
  padding: 10px 5px;
  border-radius: 4px;
  cursor: pointer;
  transition: background-color 0.2s;
}

.file-item:hover {
  background-color: var(--highlight);
  color: var(--main-bg);
}

.file-item.selected {
  background-color: var(--highlight);
  color: var(--main-bg);
}

.file-icon {
  font-size: 36px;
}
```

```
    margin-bottom: 5px;
}

.file-name {
    font-size: 12px;
    text-align: center;
    word-break: break-word;
    max-width: 100%;
}

/* Terminal */
.terminal {
    display: flex;
    flex-direction: column;
    height: 100%;
    background-color: var(--terminal-bg);
    color: var(--main-text);
    font-family: 'Courier New', monospace;
    padding: 10px;
    overflow: auto;
}

.terminal-output {
    flex: 1;
    overflow-y: auto;
    white-space: pre-wrap;
    word-break: break-all;
    line-height: 1.3;
}

.terminal-input-line {
    display: flex;
    margin-top: 10px;
}

.terminal-prompt {
    color: var(--highlight);
    margin-right: 10px;
}

.terminal-input {
    flex: 1;
    background: none;
    border: none;
    color: var(--main-text);
    font-family: 'Courier New', monospace;
    font-size: inherit;
    outline: none;
}

/* Code Editor */
.editor {
    display: flex;
    flex-direction: column;
    height: 100%;
}

.editor-toolbar {
    display: flex;
    align-items: center;
```

```
height: 40px;
padding: 0 10px;
background-color: var(--terminal-bg);
border-bottom: 1px solid var(--main-border);
}

.editor-content {
  flex: 1;
  position: relative;
}

.editor-textarea {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  padding: 10px;
  background-color: var(--terminal-bg);
  color: var(--main-text);
  border: none;
  resize: none;
  font-family: 'Courier New', monospace;
  font-size: 14px;
  line-height: 1.5;
  tab-size: 4;
  outline: none;
}

/* Settings App */
.settings {
  display: flex;
  height: 100%;
}

.settings-sidebar {
  width: 220px;
  background-color: var(--terminal-bg);
  border-right: 1px solid var(--main-border);
  overflow-y: auto;
  padding: 15px 0;
}

.settings-nav-item {
  padding: 10px 20px;
  cursor: pointer;
  transition: background-color 0.2s;
}

.settings-nav-item:hover {
  background-color: var(--highlight);
  color: var(--main-bg);
}

.settings-nav-item.active {
  background-color: var(--highlight);
  color: var(--main-bg);
}

.settings-content {
  flex: 1;
```

```
padding: 20px;
overflow-y: auto;
}
.settings-section {
    margin-bottom: 30px;
}
.settings-section-title {
    font-size: 18px;
    font-weight: 500;
    margin-bottom: 15px;
    color: var(--highlight);
}
.settings-option {
    margin-bottom: 15px;
}
.settings-label {
    display: block;
    margin-bottom: 5px;
    font-size: 14px;
}
.settings-input,
.settings-select {
    width: 100%;
    padding: 8px 10px;
    background-color: var(--terminal-bg);
    color: var(--main-text);
    border: 1px solid var(--main-border);
    border-radius: 4px;
}
/* =====
 * EFFECTS & RESPONSIVENESS
 * ===== */
/* CRT effects */
.crt-overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: linear-gradient(rgba(18, 16, 16, 0) 50%, rgba(0, 0, 0, 0.25) 50%);
    background-size: 100% 4px;
    z-index: 9999;
    opacity: 0.15;
    pointer-events: none;
    display: none; /* Toggled by user preference */
}
.scanline {
    width: 100%;
    height: 100%;
    position: fixed;
    top: 0;
    left: 0;
```

```
background: linear-gradient(to bottom,
    transparent 0%,
    rgba(var(--main-text-rgb, 51, 255, 51), 0.1) 50%,
    transparent 100%);
animation: scanline 10s linear infinite;
opacity: 0.3;
pointer-events: none;
z-index: 9998;
display: none; /* Toggled by user preference */
}

@keyframes scanline {
    0% { transform: translateY(0); }
    100% { transform: translateY(100vh); }
}

/* Responsive adjustments */
@media (max-width: 768px) {
    .desktop-icons {
        grid-template-columns: repeat(auto-fill, 70px);
        grid-gap: 10px;
    }
    .desktop-icon {
        width: 70px;
    }
    .icon-image {
        width: 40px;
        height: 40px;
        font-size: 20px;
    }
    .icon-text {
        font-size: 11px;
        max-width: 70px;
    }
    .window {
        width: 90% !important;
        height: 80% !important;
        top: 10% !important;
        left: 5% !important;
    }
    .settings-sidebar {
        width: 180px;
    }
}
```

### [33] GenesisOS/index.php

- **Bytes:** 241141
- **Type:** text

```
<?php
/**
 * Text Reversion
 * Main entry point, API dispatcher, and HTML shell.
 */

require_once 'bootstrap.php';

/* =====
 * API Dispatcher
 * ===== */
if (isset($_GET['api'])) {
    jsonHeader();

    $isAuthenticated = isAuthenticated();
    $api = $_GET['api'];
    $response = null;

    // Public endpoints that do not require authentication
    $publicEndpoints = [
        'auth' => ['login', 'register', 'validateToken'],
        'system' => ['getSystemInfo', 'getLanguagePack']
    ];

    $method = $_POST['method'] ?? $_GET['method'] ?? null;

    // Check if the requested endpoint is public
    $isPublicCall = isset($publicEndpoints[$api]) && in_array($method,
$publicEndpoints[$api]);

    if (!$isAuthenticated && !$isPublicCall) {
        echo json_encode(['success' => false, 'message' => 'Authentication
required']);
        exit;
    }

    // Route to the appropriate handler
    try {
        switch ($api) {
            case 'auth':
                $response = handleAuthAPI();
                break;
            case 'filesystem':
                $response = handleFileSystemAPI();
                break;
            case 'users':
                $response = handleUsersAPI();
                break;
            case 'system':
                $response = handleSystemAPI();
                break;
            case 'apps':
                handleAppsAPI(); // This function handles its own exit
        }
    } catch (\Exception $e) {
        $response = [
            'success' => false,
            'message' => $e->getMessage(),
            'stacktrace' => $e->getTraceAsString()
        ];
    }
}
else {
    // Handle static files
    $file = $_GET['file'];
    if (file_exists($file)) {
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment; filename=' . basename($file));
        readfile($file);
        exit;
    }
}
```

```
        break;
    case 'sandbox':
        $response = handleSandboxAPI();
        break;
    default:
        http_response_code(404);
        $response = [ 'success' => false, 'message' => 'Unknown API
endpoint'];
        break;
    }
} catch (Throwable $e) {
    error_log("API Error in {$api}: " . $e->getMessage());
    http_response_code(500);
    $response = [ 'success' => false, 'message' => 'An internal server error
occurred.'];
}
}

// The response from handlers is now directly echoed
if ($response) {
    echo json_encode($response);
}
exit;
}

// Initialize system for the main page load
initializeSystem();

// Properly close the output buffer at the end
ob_end_flush();
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Text Reversion</title>

    <!--  FIXED: Replaced inline styles with a linked stylesheet -->
    <link rel="stylesheet" href="index.css">

</head>
<body>
    <!-- CRT effects (toggled by settings) -->
    <div class="crt-overlay" id="crt-overlay"></div>
    <div class="scanline" id="scanline"></div>

    <!-- Splash Screen -->
    <div id="splash-screen" class="splash-screen">
        <div id="splash-logo">G</div>
        <div class="loading-indicator">
            <div class="loading-bar">
                <div id="loading-progress" class="loading-progress"></div>
            </div>
            <div id="loading-status">Initializing...</div>
        </div>
    </div>
```

```
</div>
</div>

<!-- Login Screen -->
<div id="login-screen" class="login-screen">
    <div class="login-container">
        <h2 class="login-heading">Text Reversion Login</h2>
        <form id="login-form" autocomplete="on">
            <div class="login-input">
                <input type="text" id="username" name="username"
placeholder="Username" required>
            </div>
            <div class="login-input">
                <input type="password" id="password" name="password"
placeholder="Password" required>
            </div>
            <button type="submit" class="login-button">Log In</button>
            <div id="login-error" class="login-error"></div>
            <div class="login-extra-links">
                <a id="register-link" style="display: none;">Register as
Developer</a>
            </div>
        </form>
    </div>
</div>

<!-- Desktop Environment -->
<div id="desktop" class="desktop">
    <div id="workspace" class="workspace">
        <!-- Desktop icons will be created here -->
        <div id="desktop-icons" class="desktop-icons"></div>

        <!-- Windows will be created here -->
    </div>
    <div id="taskbar" class="taskbar">
        <button id="start-button" class="start-button">GOS</button>
        <div id="taskbar-items" class="taskbar-items">
            <!-- Taskbar items will be created here -->
        </div>
        <div id="system-tray" class="system-tray">
            <div id="clock" class="clock"></div>
        </div>
    </div>
</div>

<!-- Start Menu -->
<div id="start-menu" class="start-menu">
    <!-- Start menu content will be generated -->
</div>

<!-- Context Menu -->
<div id="desktop-context-menu" class="context-menu" style="display: none;">
    <div class="context-menu-item" id="ctx-new-folder">New Folder</div>
    <div class="context-menu-item" id="ctx-new-file">New File</div>
```

```
<div class="context-menu-separator"></div>
<div class="context-menu-item" id="ctx-refresh">Refresh</div>
<div class="context-menu-item" id="ctx-settings">Settings</div>
</div>

<!-- Notification Container -->
<div id="notification-container" class="notification-container">
    <!-- Notifications will be created here -->
</div>

<!-- Error Banner -->
<div id="error-banner" class="error-banner"></div>

<!-- Debug Console -->
<pre id="debug-console" class="debug-console"></pre>

<!-- Error Screen -->
<div id="error-screen" class="error-screen">
    <div class="error-container">
        <h2 id="error-title">System Error</h2>
        <p id="error-message">An error occurred.</p>
        <button id="restart-button" class="button"
    onclick="window.location.reload()">Restart</button>
    </div>
</div>

<script>

/**
 * Text Reversion Kernel
 * Core system kernel for the Genesis Operating System
 */
class Kernel {
    /**
     * Create a new kernel instance
     */
    constructor() {
        // System metadata
        this.version = '1.0.0';
        this.buildDate = '2025-07-04';
        this.cacheBust = this.buildDate;
        this.codename = 'Genesis';

        // System state
        this.state = {
            status: 'uninitialized', // uninitialized -> initializing ->
running -> shuttingDown -> shutdown -> error
            startTime: null,
            lastError: null,
            debug: true,
            useLocalFilesystem: false // Will be determined during
initialization
        };
    }
}
```

```
// Core modules
this.modules = {
    events: null,      // Event management system
    security: null,   // Security and permissions
    filesystem: null, // Virtual file system
    process: null,    // Process management
    ui: null          // User interface management
};

// Runtime data
this.applications = new Map();
this.services = new Map();
this.api = {}; // <-- ADD THIS LINE
this.config = null;           // System configuration

// Translation data
this.translations = {};

// Current user context
this.currentUser = null;

// Kernel logger
this.log = this._createLogger('kernel');

    this.log.info(`Text Reversion Kernel v${this.version} created`);
}

/**
 * Initialise the kernel and all core systems
 * (fully backwards-compatible replacement for the old version)
 * @async
 * @returns {Promise<boolean>} true on success, false on fatal error
 */
async initialize() {
    try {
        /*
         * 0. Boot-strap
         */
        this._updateState('initializing');
        this.state.startTime = Date.now();
        const STEP = this._makeStepper([
            10, 'Loading system configuration...',
            20, 'Initialising core modules...',
            30, 'Initialising security...',
            40, 'Initialising filesystem...',
            50, 'Initialising process manager...',
            60, 'Connecting essential services...',
            70, 'Initialising user-interface...',
            80, 'Loading system state...',
            90, 'Finalising...',
            100, 'System ready!'
        ]);

        this.log.info('Kernel initialisation started');
    }
}
```

```
/*
 * 1. Load configuration
 *
STEP.next(); // 10 %
await this._loadConfiguration();

/* --- backward compatibility flags -----
if (this.config && typeof this.config.debug_mode === 'boolean') {
    this.state.debug = this.config.debug_mode;
    this.log.info(`Debug mode: ${this.state.debug} ? 'enabled' :
'disabled'`);
}
if (this.config && typeof this.config.use_local_filesystem ===
'boolean') {
    this.state.useLocalFilesystem =
this.config.use_local_filesystem;
    this.log.info(`Using ${this.state.useLocalFilesystem ?
'server' : 'local-storage'} filesystem`);
}

/*
 * 2. Core modules (events, security, filesystem, process)
 *
STEP.next(); // 20 %
this.modules.events = new EventSystem(this);
this._registerSystemEvents();

STEP.next(); // 30 %
this.modules.security = new SecurityManager(this);

STEP.next(); // 40 %
this.modules.filesystem = new FileSystem(this);

STEP.next(); // 50 %
this.modules.process = new ProcessManager(this);

/*
 * 3. Essential services (including "apps")
 *
STEP.next(); // 60 %
await this._connectEssentialServices();

// Load language pack
await this._loadLanguagePack();

// ----- NEW (A): ensure the Apps service exists -----
if (!this.services.has('apps')) {
    this.log.warn('Apps service not found - creating stub
(read-only)');
    this.services.register('apps', {
        async call(method, params) {
            if (method === 'getAppInfo') { return null; }
            if (method === 'listApps') { return []; }
        }
    });
}
```

```
        throw new Error(`Apps service stub: unknown method
${method}`);
    }
  });
}

// ----- ensure the service is ready -----
if (typeof this.services.get('apps').ready === 'function') {
  await this._timeout(5_000, this.services.get('apps').ready());
}

/*
 * 4. UI
 *
 */
STEP.next(); // 70 %
this.modules.ui = new UIManager(this);
await this.modules.ui.initialize();

/*
 * 5. System state (login session, etc.)
 *
 */
STEP.next(); // 80 %
const isAuthenticated = await this._loadSystemState();

/*
 * 6. Desktop or login
 *
 */
STEP.next(); // 90 %
if (isAuthenticated) {
  this.log.info('User is authenticated -- showing desktop');
  this._showDesktop();
} else {
  this.log.info('User is not authenticated -- showing login
screen');
  this._showLoginScreen();
}

/*
 * 7. Finalise
 *
 */
if (!this.state.useLocalFilesystem) {
  try {
    await this.modules.filesystem.initializeLocalStorage();
  } catch (e) {
    // ----- NEW (C): degrade gracefully -----
    this.log.warn('LocalStorage unavailable, falling back to
in-memory FS');
    await this.modules.filesystem.initializeInMemory();
  }
}

this._updateState('running');
STEP.next(); // 100 %

const initTime = ((Date.now() - this.state.startTime) /
1000).toFixed(2);
```

```
        this.log.info(`Kernel initialisation completed in ${initTime}s`);

        this.modules.events.emit('system:ready', {
            version: this.version,
            initializationTime: initTime
        });
        return true;

    } catch (error) {
        /* -----
         * Fatal error
         * -----
         */
        this.state.lastError = error;
        this._updateState('error');

        this.log.error(`Kernel initialisation failed: ${error.message}`, error);
        this._showErrorScreen('System Initialisation Failed', error.message, error.stack);
        return false;
    }
}

/* -----
 * Small helpers -- keep them private to the kernel
 * -----
 */

/** iterable progress helper (D) */
_makeStepper(pairs) {
    let idx = 0;
    return {
        next: () => {
            const pct = pairs[idx++];
            const text = pairs[idx++];
            this._updateLoadingProgress(pct, text);
        }
    };
}

/**
 * Promise with timeout (E)
 * @param {number} ms    milliseconds
 * @param {Promise} p    promise to race
 */
_timeout(ms, p) {
    return Promise.race([
        p,
        new Promise((_, reject) =>
            setTimeout(() => reject(new Error('timeout')), ms)
        )
    ]);
}
```

```
/**  
 * Load system configuration  
 * @private  
 * @async  
 */  
async _loadConfiguration() {  
    try {  
        const response = await fetch(`?api=system&method=getSystemInfo`);  
        if (!response.ok) {  
            throw new Error(`HTTP error! status: ${response.status}`);  
        }  
  
        const result = await response.json();  
        if (!result.success) {  
            throw new Error(result.message || 'Failed to fetch system  
info.');        }  
  
        this.config = result.data;  
        this.log.info('System configuration loaded successfully.');  
  
        //  HARDED & CORRECT: Use optional chaining to safely access  
nested properties.  
        const allowRegistration =  
this.config?.security?.allow_developer_registration;  
  
        if (allowRegistration) {  
            const registerLink = document.getElementById('register-link');  
            if (registerLink) {  
                registerLink.style.display = 'inline';  
                registerLink.onclick = () =>  
this.modules.ui.showRegistrationDialog();  
            }  
        }  
  
        return true;  
    } catch (error) {  
        this.log.error('Failed to load system configuration', error);  
        throw new Error(`Configuration loading failed: ${error.message}`);  
    }  
}  
  
/**  
 * Launch an application  
 * @async  
 * @param {string} appId - Application identifier  
 * @param {Object} params - Launch parameters  
 * @returns {Promise<string>} Process ID of the launched application  
 */  
async launchApplication(appId, params = {}) {  
    try {  
        this.log.info(`Launching application: ${appId}`);  
  
        /* ----- */  
    } catch (error) {  
        this.log.error(`Failed to launch application: ${appId}`, error);  
        throw new Error(`Application launching failed: ${error.message}`);  
    }  
}
```

```
* 1. Permission check
* -----
if (!this.modules.security.hasPermission(`app.launch.${appId}`)) {
    this.modules.ui.showNotification(
        'Permission Denied',
        `You don't have permission to launch ${appId}`,
        5000
    );
    throw new Error(`Permission denied: Cannot launch ${appId}`);
}

/*
* 2. Fetch manifest      (note the { id : appId } param)
* -----
const appInfo = await this.services
    .get('apps')
    .call('getAppInfo', { id: appId });

if (!appInfo) {
    throw new Error(`Application not
found: ${appId}`);
}

if (Array.isArray(appInfo.permissions)) {
    for (const perm of
appInfo.permissions) {
        if
(!this.modules.security.hasPermission(perm)) {
            this.modules.ui.showNotification(
                'Permission
Denied',
                `Missing
permission ${perm} for ${appId}`,
                5000
            );
            throw new
Error(`Permission denied: ${perm}`);
        }
    }
}

/*
* 3. Create a process / window
* -----
const process = await this.modules.process.createProcess(
    appId,
    appInfo,
    params
);

/*
* 4. Resolve the entry point and execute it
* -----

```

```
        await this._bootApplicationCode(appInfo, process);

        /*
         * 5. Book-keeping & events
         */
        this.applications.set(process.id, process);

        this.modules.events.emit('application:launched', {
            appId,
            processId: process.id,
            windowId: process.windowId
        });

        return process.id;
    } catch (error) {
        this.log.error(`Failed to launch application ${appId}:`, error);

        this.modules.events.emit('application:launchFailed', {
            appId,
            error: error.message
        });

        this.modules.ui.showNotification(
            'Application Error',
            `Failed to launch ${appId}: ${error.message}`,
            5000
        );

        throw error;
    }
}

/**
 * Connect to essential backend services
 * @private
 * @async
 */
async _connectEssentialServices() {
    try {
        // Connect to essential services
        const essentialServices = ['auth', 'users', 'apps', 'filesystem',
'system', 'sandbox'];

        for (const serviceName of essentialServices) {
            this.log.info(`Connecting to ${serviceName} service...`);
            const service = this._createServiceProxy(serviceName);

            // Store the service in both locations
            this.services.set(serviceName, service);
            this.api[serviceName] = service; // <-- ADD THIS LINE
        }

        this.log.info('All essential services connected');
        return true;
    }
}
```

```
        } catch (error) {
            this.log.error('Failed to connect to essential services', error);
            throw new Error(`Service connection failed: ${error.message}`);
        }
    }

/**
 * Create a service proxy for API calls
 * @private
 * @param {string} serviceName - Name of the service
 * @returns {Object} Service proxy object
 */
_createServiceProxy(serviceName) {
    const kernel = this;

    return {
        name: serviceName,
        status: 'connected',

        /**
         * Call a method on the service
         * @async
         * @param {string} method - Method name
         * @param {Object} params - Method parameters
         * @returns {Promise<any>} Response data
         */
        async call(method, params = {}) {
            try {
                const requestId = kernel._generateRequestId();

                // ✅ FIXED: Switched from FormData to a standard JSON
                body to prevent 403 errors from server security rules.
                const body = JSON.stringify({ method, params });
                const response = await fetch(`index.php?
api=${this.name}&v=${kernel.cacheBust}&_=${Date.now()}`, {
                    method: 'POST',
                    headers: {
                        'X-GOS-Request-ID': requestId,
                        'Content-Type': 'application/json'
                    },
                    body: body,
                    credentials: 'same-origin'
                });

                // Rest of the function remains the same...

                // Check for HTTP errors
                if (!response.ok) {
                    throw new Error(`Service returned status
${response.status} ${response.statusText}`);
                }

                // Parse response
                const result = await response.json();
            }
        }
}
```

```
// Check for API errors
if (result.success === false) {
    throw new Error(result.message || 'Unknown service
error');
}

return result.data;
} catch (error) {
    kernel.log.error(`Service ${this.name}.${method} call
failed:`, error);
    throw error;
}
};

}

/***
 * Load saved system state and check authentication
 * @private
 * @async
 * @returns {Promise<boolean>} True if user is authenticated
 */
async _loadSystemState() {
    try {
        // Check if user is authenticated
        if (this.services.has('auth')) {
            this.log.info('Checking authentication state...');

            const authService = this.services.get('auth');
            const validationResult = await
authService.call('validateToken', {});

            if (validationResult.valid && validationResult.user) {
                this.log.info(`Valid session found for user:
${validationResult.user.username}`);

                // Set current user
                this.currentUser = validationResult.user;

                // Update security context with user permissions
                this.modules.security.handleLogin({
                    user: validationResult.user,
                    permissions: validationResult.permissions || []
                });

                // Emit user login event
                this.modules.events.emit('auth:restored', {
                    user: validationResult.user
                });

                return true;
            } else {
                this.log.info('No valid authentication session found');
            }
        }
    } catch (error) {
        this.log.error(`Error loading system state: ${error}`);
        throw error;
    }
}
}
```

```
        }

    }

    return false;
} catch (error) {
    this.log.error('Failed to load system state', error);
    // Don't throw here, as we can continue without a restored session
    return false;
}

}

/***
 * Register handlers for critical system events
 * @private
 */
_registerSystemEvents() {
    // Handle authentication events
    this.modules.events.on('auth:login', (data) => {
        this.log.info(`User logged in: ${data.user.username}`);
        this.currentUser = data.user;

        // Update security context
        this.modules.security.handleLogin(data);

        // Show desktop
        this._showDesktop();
    });

    this.modules.events.on('auth:logout', () => {
        this.log.info('User logged out');
        this.currentUser = null;

        // Update security context
        this.modules.security.handleLogout();

        // Show login screen
        this._showLoginScreen();
    });

    // Handle system shutdown event
    this.modules.events.on('system:shutdown', (data) => {
        this.log.info(`System shutdown initiated: ${data.reason}`);
        this.shutdown(data.reason);
    });

    // Handle application lifecycle events
    this.modules.events.on('application:launched', (data) => {
        this.log.info(`Application launched: ${data.appId} (Process ID: ${data.processId})`);
    });

    this.modules.events.on('application:terminated', (data) => {
        this.log.info(`Application terminated: ${data.appId} (Process ID: ${data.processId})`);
    });
}
```

```
// Remove from running applications
this.applications.delete(data.processId);
});

}

/**
 * Update the loading progress during initialization
 * @private
 * @param {number} progress - Progress percentage (0-100)
 * @param {string} status - Status message
 */
_updateLoadingProgress(progress, status) {
    const progressBar = document.getElementById('loading-progress');
    const statusText = document.getElementById('loading-status');

    if (progressBar) {
        progressBar.style.width = `${progress}%`;
    }

    if (statusText && status) {
        statusText.textContent = status;
    }
}

_showLoginScreen() {
    const splash = document.getElementById('splash-screen');
    const login = document.getElementById('login-screen');
    splash.classList.add('fade-out');

    setTimeout(() => {
        splash.style.display = 'none';
        login.style.display = 'flex';
        setTimeout(() => login.classList.add('show'), 50);
    }, 500);

    // ✅ FIXED: This logic now correctly runs every time the login screen is shown.
    const allowRegistration =
this.config?.security?.allow_developer_registration;
    const registerLink = document.getElementById('register-link');

    if (allowRegistration && registerLink) {
        // Only show the link if no user is currently logged in.
        // This handles the case where a user logs out and returns to this screen.
        if (!this.currentUser) {
            registerLink.style.display = 'inline';
            registerLink.onclick = () =>
this.modules.ui.showRegistrationDialog();
        } else {
            registerLink.style.display = 'none';
        }
    } else if (registerLink) {

```

```
        registerLink.style.display = 'none';
    }

    const form = document.getElementById('login-form');
    form.onsubmit = async (e) => {
        e.preventDefault();
        const errorDiv = document.getElementById('login-error');
        errorDiv.textContent = '';
        try {
            const result = await this.api.auth.call('login', { username:
form.username.value, password: form.password.value });
            this.modules.events.emit('auth:login', { user: result.user,
permissions: result.permissions });
            login.classList.remove('show');
            setTimeout(() => login.style.display = 'none', 500);
        } catch (error) {
            errorDiv.textContent = error.message;
        }
    };
}

/**
 * Show the desktop environment
 * @private
 */
_showDesktop() {
    // Hide splash screen
    const splashScreen = document.getElementById('splash-screen');
    splashScreen.classList.add('fade-out');

    // Hide login screen if visible
    const loginScreen = document.getElementById('login-screen');
    loginScreen.classList.remove('show');

    setTimeout(() => {
        splashScreen.style.display = 'none';
        loginScreen.style.display = 'none';

        // Show desktop
        const desktop = document.getElementById('desktop');
        desktop.style.display = 'block';

        setTimeout(() => {
            desktop.classList.add('show');
        }, 50);

        // Create desktop icons
        this._createDesktopIcons();

        // Update the clock
        this._updateClock();
        setInterval(() => this._updateClock(), 60000);

        // Set up start button
    }, 50);
}
```

```
document.getElementById('start-button').addEventListener('click',
() => {
    this.toggleStartMenu();
});

// Set up desktop context menu
const desktopElement = document.getElementById('desktop');
desktopElement.addEventListener('contextmenu', (e) => {
    // Only show context menu on desktop background or icons
    container
    if (e.target === desktopElement || e.target.id === 'desktop-
icons') {
        e.preventDefault();
        this._showContextMenu(e.pageX, e.pageY);
    }
});

// Close context menu when clicking elsewhere
document.addEventListener('click', () => {
    document.getElementById('desktop-context-menu').style.display
= 'none';
});

// Set up context menu items
document.getElementById('ctx-new-
folder').addEventListener('click', () => {
    this._createNewFolder();
});

document.getElementById('ctx-new-file').addEventListener('click',
() => {
    this._createNewFile();
});

document.getElementById('ctx-refresh').addEventListener('click',
() => {
    this._refreshDesktop();
});

document.getElementById('ctx-settings').addEventListener('click',
() => {
    this.launchApplication('settings');
});

// Show welcome notification
this.modules.ui.showNotification(
    'Welcome to Text Reversion',
    `Hello, ${this.currentUser.name}! Welcome to Text Reversion
v${this.version}.`,
    5000
);
}, 500);
}
```

```
_createDesktopIcons() {
    const desktopIcons = document.getElementById('desktop-icons');
    desktopIcons.innerHTML = '';

    // Log the start of the process for debugging
    this.log.debug('Creating desktop icons');

    // Request the full app list from the service. This is the single
    source of truth.
    this.services.get('apps').call('listApps').then(allApps => {

        // Filter for built-in apps only. External/filesystem apps have a
        `__path` property,
        // while built-in apps do not. This prevents external apps from
        cluttering the desktop.
        const builtInApps = allApps.filter(app => !app.__path);

        this.log.debug(`Found ${builtInApps.length} built-in apps for
        desktop icons.`);

        // Create an icon for each built-in app that is marked for desktop
        display.
        builtInApps.forEach(app => {
            if (app.desktopIcon !== false) {
                const iconElement = document.createElement('div');
                iconElement.className = 'desktop-icon';
                iconElement.dataset.appId = app.id;

                const iconImage = document.createElement('div');
                iconImage.className = 'icon-image';
                iconImage.textContent = app.icon ||
                app.id.charAt(0).toUpperCase();

                const iconText = document.createElement('div');
                iconText.className = 'icon-text';
                iconText.textContent = app.title;

                iconElement.appendChild(iconImage);
                iconElement.appendChild(iconText);

                // Add click handler to launch the application.
                iconElement.addEventListener('click', () => {
                    this.launchApplication(app.id);
                });

                // Add context menu handler (currently does nothing).
                iconElement.addEventListener('contextmenu', (e) => {
                    e.preventDefault();
                    // Custom context menu for desktop icons could be
                    added here.
                });

                desktopIcons.appendChild(iconElement);
            }
        })
    })
}
```

```
        });
    }).catch(error => {
      this.log.error('Failed to load app list for desktop icons:',
error);

      // Show error notification to the user.
      this.modules.ui.showNotification(
        'Error',
        'Failed to load desktop applications',
        5000
      );
    });
}

/**
 * Show the context menu
 * @private
 * @param {number} x - X position
 * @param {number} y - Y position
 */
_showContextMenu(x, y) {
  const contextMenu = document.getElementById('desktop-context-menu');
  contextMenu.style.left = `${x}px`;
  contextMenu.style.top = `${y}px`;
  contextMenu.style.display = 'block';
}

/**
 * Create a new folder on desktop
 * @private
 */
_createNewFolder() {
  const folderName = prompt('Enter folder name:');
  if (!folderName) return;

  const path =
`/users/${this.currentUser.username}/Desktop/${folderName}`;

  this.modules.filesystem.createDirectory(path).then(() => {
    this.modules.ui.showNotification(
      'Folder Created',
      `Created folder: ${folderName}`,
      3000
    );
    this._refreshDesktop();
}).catch(error => {
  this.modules.ui.showNotification(
    'Error',
    `Failed to create folder: ${error.message}`,
    5000
  );
});
}
}
```

```
/**  
 * Create a new file on desktop  
 * @private  
 */  
_createNewFile() {  
    const fileName = prompt('Enter file name:');  
    if (!fileName) return;  
  
    const path =  
`/users/${this.currentUser.username}/Desktop/${fileName}`;  
  
    this.modules.filesystem.writeFile(path, '') .then(() => {  
        this.modules.ui.showNotification(  
            'File Created',  
            `Created file: ${fileName}`,  
            3000  
        );  
        this._refreshDesktop();  
    }).catch(error => {  
        this.modules.ui.showNotification(  
            'Error',  
            `Failed to create file: ${error.message}`,  
            5000  
        );  
    });  
}  
  
/**  
 * Refresh the desktop  
 * @private  
 */  
_refreshDesktop() {  
    this._createDesktopIcons();  
    this.modules.ui.showNotification(  
        'Desktop Refreshed',  
        'The desktop has been refreshed',  
        2000  
    );  
}  
  
/**  
 * Show the error screen  
 * @private  
 * @param {string} title - Error title  
 * @param {string} message - Error message  
 * @param {string} details - Error details (stack trace)  
 */  
_showErrorScreen(title, message, details) {  
    // Hide other screens  
    document.getElementById('splash-screen').style.display = 'none';  
    document.getElementById('login-screen').style.display = 'none';  
    document.getElementById('desktop').style.display = 'none';  
  
    // Set error information
```

```
document.getElementById('error-title').textContent = title;
document.getElementById('error-message').textContent = message;

const errorDetails = document.getElementById('error-details');
if (details) {
    errorDetails.textContent = details;
    errorDetails.style.display = 'block';
}

// Show error screen
document.getElementById('error-screen').style.display = 'flex';

// Set up restart button
document.getElementById('restart-button').onclick = () => {
    window.location.reload();
};

/***
 * Update the clock element
 * @private
 */
_updateClock() {
    const clock = document.getElementById('clock');
    if (!clock) return;

    const now = new Date();
    const hours = now.getHours().toString().padStart(2, '0');
    const minutes = now.getMinutes().toString().padStart(2, '0');
    clock.textContent = `${hours}:${minutes}`;
}

/***
 * Toggle the start menu visibility
 */
toggleStartMenu() {
    // Check if start menu already exists and is populated
    let startMenu = document.getElementById('start-menu');

    if (startMenu.classList.contains('active')) {
        startMenu.classList.remove('active');
        return;
    }

    // Populate the start menu
    this._populateStartMenu();

    // Show the menu
    startMenu.classList.add('active');

    // Close when clicking outside
    document.addEventListener('click', function closeMenu(e) {
        if (!startMenu.contains(e.target) && e.target.id !== 'start-
button') {

```

```
        startMenu.classList.remove('active');
        document.removeEventListener('click', closeMenu);
    }
});

}

/**
 * Populate the start menu with content.
 *  MODIFIED: Now includes a search bar and moves the logout button.
 * @private
 */
_populateStartMenu() {
    const startMenu = document.getElementById('start-menu');
    const userName = this.currentUser ? this.currentUser.name : 'Guest';

    // --- 1. Create the new HTML structure ---
    startMenu.innerHTML =
        `<div class="user-info" style="display: flex; justify-content: space-between; align-items: center;">
            <div style="display: flex; align-items: center;">
                <div class="user-avatar">${userName.charAt(0)}</div>
                <div class="user-name">${userName}</div>
            </div>
            <div class="menu-item" data-action="logout" title="Log Out" style="padding: 5px 10px; margin-right: 10px;">
                <div class="menu-item-icon">Logout</div>
            </div>
        </div>
        <div class="start-menu-search-box" style="padding: 5px 10px;">
            <input type="text" id="start-menu-search" placeholder="Search applications..." style="width: 100%; padding: 8px; background: var(--terminal-bg); border: 1px solid var(--main-border); color: var(--main-text); border-radius: 3px;">
        </div>
        <div class="menu-items"></div>
    `;
}

const menuItemsContainer = startMenu.querySelector('.menu-items');

// --- 2. Fetch and render app list ---
this.services.get('apps').call('listApps').then(apps => {
    let menuHtml = '';
    apps.forEach(app => {
        menuHtml += `
            <div class="menu-item" data-app="${app.id}">
                <div class="menu-item-icon">${app.icon} || ${app.title}</div>
            </div>
        `;
    });
    menuItemsContainer.innerHTML = menuHtml;

    // --- 3. Add Event Listeners ---
    menuItemsContainer.querySelectorAll('.menu-item[data-')
}
```

```
app'].forEach(item => {
    item.addEventListener('click', () => {
        this.launchApplication(item.getAttribute('data-app'));
        this.toggleStartMenu();
    });
});

// Logout button
startMenu.querySelector('.menu-item[data-action="logout"]').addEventListener('click', () => {
    this.logout();
});

// Search functionality
document.getElementById('start-menu-search').addEventListener('input', (e) => {
    const searchTerm = e.target.value.toLowerCase();
    menuItemsContainer.querySelectorAll('.menu-item[data-app]').forEach(item => {
        const appName = item.textContent.toLowerCase();
        if (appName.includes(searchTerm)) {
            item.style.display = 'flex';
        } else {
            item.style.display = 'none';
        }
    });
});

}).catch(error => {
    this.log.error('Failed to load app list for start menu', error);
    menuItemsContainer.innerHTML = `<div class="menu-item">Error loading applications</div>`;
});

}

/**
 * Load the application's entry point and hand over control.
 *  MODIFIED: Now uses the `getAppAsset` API endpoint to securely load module scripts.
 * @private
 * @param {Object} appInfo -- manifest from getAppInfo
 * @param {Object} process -- result of createProcess()
 */
async _bootApplicationCode(appInfo, process) {
    const entry = appInfo.entry || '';

    if (appInfo._path && entry.endsWith('.js')) {
        const windowContent = document.querySelector(`#${process.windowId} .window-content`);
        const iframe = document.createElement('iframe');
        //iframe.setAttribute('sandbox', 'allow-scripts allow-same-origin');
        //  FIXED: Added 'allow-downloads', 'allow-popups', and 'allow-forms' to the sandbox.
    }
}
```

```
// This resolves the download error and increases compatibility
for applications.

    iframe.setAttribute('sandbox', 'allow-scripts allow-same-origin
allow-forms allow-popups allow-downloads');
    iframe.style.border = 'none';
    iframe.style.width = '100%';
    iframe.style.height = '100%';
    windowContent.innerHTML = '';
    windowContent.appendChild(iframe);

// in index.php -> _bootApplicationCode method

const gosApiProxy = {
    app: { id: appInfo.id, title: appInfo.title, version:
appInfo.version, params: process.params },
    //  ADDED: Expose the current user's data to the sandboxed
application.
    user: {
        username: this.currentUser.username,
        name: this.currentUser.name,
        roles: this.currentUser.roles
    },
    window: {
        getContainer: () => iframe.contentDocument.body,
        setTitle: (newTitle) => {
            const titleEl =
document.querySelector(`#${process.windowId} .window-title`);
            if (titleEl) titleEl.textContent = newTitle;
        },
        close: () =>
this.modules.process.terminateProcess(process.id),
    },
    filesystem: this.modules.filesystem,
    ui: this.modules.ui,
    events: this.modules.events,
};

iframe.addEventListener('load', () => {
    try {
        const scriptUrl =
`//${window.location.host}${window.location.pathname}?
api=apps&method=getAppAsset&appId=${appInfo.id}&file=${entry}&v=${this.cacheBust}`;
;

        //  SIMPLIFIED: Create a global map on the parent
window to hold API proxies.
        // This is a more robust way to bridge the sandbox than
deep-path traversal.
        if (!window.gosApiProxies) {
            window.gosApiProxies = {};
        }
        window.gosApiProxies[process.id] = gosApiProxy;

        const script =
```

```
iframe.contentDocument.createElement('script');
    script.type = 'module';
    script.innerHTML = `
        try {
            const module = await import("${scriptUrl}");
            if (module.initialize && typeof module.initialize
=== 'function') {
                // ✅ SIMPLIFIED: Retrieve the API proxy from
the new global map.
                const apiProxy =
window.parent.gosApiProxies[$process.id];
                module.initialize(apiProxy);
            } else {
                throw new Error("Application entry point must
export an 'initialize' function.");
            }
        } catch(e) {
            document.body.innerHTML = '<div
style="padding:20px;color:red;"><h4>App Error</h4><p>' + e.message + '</p></div>';
            console.error('Error within sandboxed app:', e);
        }
    `;
}

iframe.contentDocument.body.appendChild(script);

} catch (e) {
    this.log.error(`Failed to initialize sandboxed app
${appInfo.id}:`, e);
    iframe.contentDocument.body.innerHTML = `<div
style="padding:20px;color:red;"><h4>App Error</h4><p>${e.message}</p></div>`;
}
});

iframe.src = 'sandbox.html';
return;
}

// --- Handle Built-in System Applications ---
else if (typeof window[entry] === 'function') {
    await window[entry](process);
    return;
}

throw new Error(`Cannot resolve entry point "${entry}" for app
${appInfo.id}`);
}

/**
 * Load language pack
 * @private
 * @async
 */
async _loadLanguagePack() {
    try {
```

```
        const systemService = this.services.get('system');
        const lang = this.config?.ui?.language || 'en';
        const result = await systemService.call('getLanguagePack', {
lang });
            this.translations = result || {};
            return true;
        } catch (error) {
            this.log.warn('Failed to load language pack', error);
            this.translations = {};
            return false;
        }
    }

/**
 * Translate a key using loaded language pack
 * @param {string} key
 * @returns {string}
 */
translate(key) {
    return this.translations[key] || key;
}

/**
 * Log out the current user
 * @async
 */
async logout() {
    try {
        this.log.info('Logging out user...');

        // Call logout API
        if (this.services.has('auth')) {
            await this.services.get('auth').call('logout', {});
        }

        // Close all running applications
        for (const [processId, process] of this.applications.entries()) {
            this.log.info(`Terminating process: ${processId}`);
            await this.modules.process.terminateProcess(processId);
        }

        // Clear current user
        this.currentUser = null;

        // Update security context
        this.modules.security.handleLogout();

        // Emit logout event
        this.modules.events.emit('auth:logout', {});
    } catch (error) {
        this.log.error('Logout failed:', error);
        throw error;
    }
}
```

```
/**
 * Shut down the system
 * @async
 * @param {string} reason - Reason for shutdown
 */
async shutdown(reason = 'user_initiated') {
    try {
        this.log.info(`Initiating system shutdown (Reason: ${reason})...`);
        this._updateState('shuttingDown');

        // Emit pre-shutdown event
        this.modules.events.emit('system:shuttingDown', { reason });

        // Close all running applications
        for (const [processId, process] of this.applications.entries()) {
            this.log.info(`Terminating process: ${processId}`);
            await this.modules.process.terminateProcess(processId);
        }

        // Log out user if logged in
        if (this.currentUser) {
            await this.logout();
        }

        // Disconnect from all services
        for (const [serviceName, service] of this.services.entries()) {
            this.log.info(`Disconnecting from service: ${serviceName}`);
            service.status = 'disconnected';
        }

        // Final shutdown event
        this.modules.events.emit('system:shutdown', { reason });

        this._updateState('shutdown');
        this.log.info('System shutdown complete');

        // Show shutdown message
        document.body.innerHTML = `
            <div style="display:flex;justify-content:center;align-items:center;height:100vh;font-size:24px;background-color:var(--main-bg);color:var(--main-text);">
                <div style="text-align:center;">
                    <p>Text Reversion has been shut down.</p>
                    <p style="font-size:16px;margin-top:20px;opacity:0.7;">Refresh the page to restart.</p>
                </div>
            </div>
        `;

        return true;
    } catch (error) {
        this.log.error('Shutdown failed:', error);
    }
}
```

```
        this._updateState('error');
        throw error;
    }
}

/**
 * Create a logger for a specific component
 * @private
 * @param {string} component - Component name
 * @returns {Object} Logger object
 */
_createLogger(component) {
    return {
        /**
         * Log an informational message
         * @param {string} message - Message to log
         * @param {Object} [data] - Additional data to log
         */
        info: (message, data) => {
            if (!this.state.debug && !component.startsWith('kernel'))
return;
            console.info(`[GOS:${component}] ${message}`, data || '');
        },
        /**
         * Log a warning message
         * @param {string} message - Message to log
         * @param {Object} [data] - Additional data to log
         */
        warn: (message, data) => {
            console.warn(`[GOS:${component}] ${message}`, data || '');
        },
        /**
         * Log an error message
         * @param {string} message - Message to log
         * @param {Error|Object} [error] - Error object or data
         */
        error: (message, error) => {
            console.error(`[GOS:${component}] ${message}`, error || '');
        },
        /**
         * Log a debug message (only shown in debug mode)
         * @param {string} message - Message to log
         * @param {Object} [data] - Additional data to log
         */
        debug: (message, data) => {
            if (!this.state.debug) return;
            console.debug(`[GOS:${component}] ${message}`, data || '');
        }
    };
}
```

```
/**  
 * Update the kernel state  
 * @private  
 * @param {string} status - New status  
 */  
_updateState(status) {  
    const previous = this.state.status;  
    this.state.status = status;  
    this.log.info(`Kernel state updated: ${status}`);  
  
    // Emit state change event if events module is available  
    if (this.modules.events) {  
        this.modules.events.emit('kernel:stateChanged', {  
            previousStatus: previous,  
            currentStatus: status  
        });  
    }  
}  
  
/**  
 * Generate a unique request ID  
 * @private  
 * @returns {string} Unique ID  
 */  
  
_generateRequestId() {  
    return 'req_' + Math.random().toString(36).substring(2, 15) +  
        Math.random().toString(36).substring(2, 15);  
}  
  
/**  
 * Dynamically load a script once  
 * @private  
 * @param {string} url - Script URL  
 * @returns {Promise<void>}  
 */  
_lazyLoadScript(url) {  
    if (document.querySelector(`script[data-src="${url}"]`)) {  
        return Promise.resolve();  
    }  
    return new Promise((resolve, reject) => {  
        const s = document.createElement('script');  
        s.type = 'module';  
        s.dataset.src = url;  
        s.onload = () => resolve();  
        s.onerror = () => reject(new Error('Failed to load  
' + url));  
        s.src = url + (url.includes('?') ? '&' : '?') + 'v=' +  
this.cacheBust;  
        document.body.appendChild(s);  
    });  
}  
/**
```

```
* Get system information
* @returns {Object} System information
*/
getSystemInfo() {
    return {
        name: this.config ? this.config.name : 'Text Reversion',
        version: this.version,
        buildDate: this.buildDate,
        codename: this.codename,
        status: this.state.status,
        uptime: this.state.startTime ? Math.floor((Date.now() - this.state.startTime) / 1000) : 0,
        user: this.currentUser ? {
            username: this.currentUser.username,
            name: this.currentUser.name,
            roles: this.currentUser.roles // <-- CORRECTED
        } : null,
        debug: this.state.debug,
        useLocalFilesystem: this.state.useLocalFilesystem,
        modules: Object.fromEntries(
            Object.entries(this.modules)
                .filter(([_, module]) => module !== null)
                .map(([name, _]) => [name, true])
        ),
        services: Array.from(this.services.keys()),
        applications: this.applications.size
    };
}

/**
 * Event System
 * Manages event subscriptions and notifications
 */
class EventSystem {
    /**
     * Create a new event system
     * @param {Kernel} kernel - Kernel reference
     */
    constructor(kernel) {
        this.kernel = kernel;
        this.events = new Map();
        this.log = kernel._createLogger('events');

        this.log.info('Event system initialized');
    }

    /**
     * Register an event listener
     * @param {string} eventName - Event name to listen for
     * @param {Function} handler - Event handler function
     * @returns {EventSystem} This event system for chaining
     */
    on(eventName, handler) {
```

```
    if (!this.events.has(eventName)) {
        this.events.set(eventName, []);
    }

    this.events.get(eventName).push(handler);
    this.log.debug(`Registered handler for event: ${eventName}`);

    return this;
}

/**
 * Remove an event listener
 * @param {string} eventName - Event name
 * @param {Function} [handler] - Event handler (if not provided, all
handlers are removed)
 * @returns {EventSystem} This event system for chaining
 */
off(eventName, handler) {
    if (!this.events.has(eventName)) return this;

    if (!handler) {
        // Remove all handlers
        this.events.delete(eventName);
        this.log.debug(`Removed all handlers for event: ${eventName}`);
    } else {
        // Remove specific handler
        const handlers = this.events.get(eventName);
        const index = handlers.indexOf(handler);

        if (index !== -1) {
            handlers.splice(index, 1);
            this.log.debug(`Removed handler for event: ${eventName}`);
        }
    }

    if (handlers.length === 0) {
        this.events.delete(eventName);
    }
}

return this;
}

/**
 * Emit an event
 * @param {string} eventName - Event name to emit
 * @param {Object} data - Event data
 */
emit(eventName, data = {}) {
    if (!this.events.has(eventName)) {
        this.log.debug(`No handlers for event: ${eventName}`);
        return;
    }

    this.log.debug(`Emitting event: ${eventName}`, data);
}
```

```
        for (const handler of this.events.get(eventName)) {
            try {
                handler(data);
            } catch (error) {
                this.log.error(`Error in event handler for ${eventName}:`, error);
            }
        }
    }

/**
 * Security Manager
 * Handles user authentication, permissions, and security policies.
 * This version is updated to support a multi-role user system.
 */
class SecurityManager {
    /**
     * Create a new security manager
     * @param {Kernel} kernel - Kernel reference
     */
    constructor(kernel) {
        this.kernel = kernel;
        this.log = kernel._createLogger('security');
        this.permissions = new Set();
        this.user = null; // This will store the full user object on login

        this.log.info('Security manager initialized');
    }

    /**
     * Handle user login. This method is called by the Kernel's event system.
     * @param {Object} data - Login data containing the user object and their permissions.
     */
    handleLogin(data) {
        this.user = data.user;

        // Set permissions from the server
        this.permissions.clear();
        if (Array.isArray(data.permissions)) {
            data.permissions.forEach(permission => {
                this.permissions.add(permission);
            });
        }

        this.log.info(`User ${this.user.username} logged in with ${this.permissions.size} permissions`);
    }

    /**
     * Handle user logout.
     */
}
```

```
handleLogout() {
    this.user = null;
    this.permissions.clear();
    this.log.info('User logged out, permissions cleared');
}

/**
 * NEW: Checks if the current user has the 'admin' role.
 * This is the key function needed by the App Store UI to show the
Submissions tab.
 * @returns {boolean} True if the user is an administrator.
 */
isAdmin() {
    // Correctly checks if the user object exists and its 'roles' array
includes 'admin'.
    return this.user && Array.isArray(this.user.roles) &&
this.user.roles.includes('admin');
}

/**
 * Check if the current user has a specific permission.
 * @param {string} permission - The permission to check (e.g.,
'process.exec').
 * @returns {boolean} True if the user has the permission.
 */
hasPermission(permission) {
    if (!this.user) {
        return false;
    }

    // UPDATED: Use the new isAdmin() method for the check.
    // This makes the permission check compatible with the multi-role
system.
    if (this.isAdmin()) {
        return true; // Admins have all permissions.
    }

    // Check for direct permission
    if (this.permissions.has(permission)) {
        return true;
    }

    // Check for wildcard permissions (e.g., app.launch.*)
    const parts = permission.split('.');
    while (parts.length > 1) {
        parts.pop();
        const wildcard = parts.join('.') + '.*';
        if (this.permissions.has(wildcard)) {
            return true;
        }
    }

    return false;
}
```

```
/**
 * Get the current user's permissions as an array.
 * @returns {Array<string>} A list of the user's permissions.
 */
getUserPermissions() {
    return Array.from(this.permissions);
}

/**
 * File System
 * Provides a virtual file system interface
 */
class FileSystem {
    /**
     * Create a new file system
     * @param {Kernel} kernel - Kernel reference
     */
    constructor(kernel) {
        this.kernel = kernel;
        this.log = kernel._createLogger('filesystem');

        this.log.info('File system initialized');
    }

    /**
     * Initialize LocalStorage-based filesystem
     * @async
     */
    async initializeLocalStorage() {
        // Create basic directory structure if it doesn't exist
        const basicDirs = [
            '/',
            '/users',
            `/users/${this.kernel.currentUser.username}`,
            `/users/${this.kernel.currentUser.username}/Desktop`,
            `/users/${this.kernel.currentUser.username}/Documents`,
            '/apps',
            '/system'
        ];

        // Check if filesystem is initialized
        if (!localStorage.getItem('gos_filesystem')) {
            this.log.info('Initializing localStorage filesystem');

            // Create initial filesystem structure
            const fs = {};

            basicDirs.forEach(dir => {
                fs[dir] = {
                    type: 'directory',
                    children: [],
                    created: Date.now(),

```

```
        modified: Date.now()
    };

    // Add to parent
    if (dir !== '/') {
        const parentDir = dir.substring(0, dir.lastIndexOf('/'))
|| '/';
        const dirName = dir.substring(dir.lastIndexOf('/') + 1);

        if (fs[parentDir] &&
!fs[parentDir].children.includes(dirName)) {
            fs[parentDir].children.push(dirName);
        }
    }
});

// Save filesystem
localStorage.setItem('gos_filesystem', JSON.stringify(fs));

// Create welcome file
await
this.writeFile(`~/users/${this.kernel.currentUser.username}/Desktop/welcome.txt`,
`Welcome to Text Reversion!\n\nThis is your personal desktop.
You can create files and folders here.\n\nEnjoy exploring the system!`);

    this.log.info('localStorage filesystem initialized');
}
}

/**
 * Check if a file exists
 * @param {string} path - Virtual file path
 * @returns {boolean} True if file exists
 */
fileExists(path) {
    if (this.kernel.state.useLocalFilesystem) {
        // Use server filesystem - call would go here
        // For now, we'll just try to read the file and catch errors
        try {
            this.readFile(path);
            return true;
        } catch (error) {
            return false;
        }
    } else {
        // Use localStorage-based filesystem
        const fs = this._getLocalFilesystem();
        return fs[path] !== undefined && fs[path].type === 'file';
    }
}

/**
 * Check if a directory exists
 * @param {string} path - Virtual directory path

```

```
* @returns {boolean} True if directory exists
*/
directoryExists(path) {
    if (this.kernel.state.useLocalFilesystem) {
        // Use server filesystem - call would go here
        // For now, we'll just try to list the directory and catch errors
        try {
            this.listDirectory(path);
            return true;
        } catch (error) {
            return false;
        }
    } else {
        // Use localStorage-based filesystem
        const fs = this._getLocalFilesystem();
        return fs[path] !== undefined && fs[path].type === 'directory';
    }
}

/**
 * Read a file
 * @async
 * @param {string} path - Virtual file path
 * @returns {Promise<Object>} File content and metadata
 */
async readFile(path) {
    try {
        this.log.debug(`Reading file: ${path}`);

        // Check permission
        if
(!this.kernel.modules.security.hasPermission('filesystem.read.*')) {
            throw new Error('Permission denied');
        }

        if (this.kernel.state.useLocalFilesystem) {
            // Call filesystem service
            const result = await
this.kernel.services.get('filesystem').call('readFile', { path });
            return result;
        } else {
            // Use localStorage-based filesystem
            const fs = this._getLocalFilesystem();

            if (!fs[path] || fs[path].type !== 'file') {
                throw new Error('File not found');
            }

            return {
                content: fs[path].content,
                size: fs[path].content.length,
                modified: fs[path].modified
            };
        }
    }
}
```

```
        } catch (error) {
            this.log.error(`Failed to read file ${path}:`, error);
            throw error;
        }
    }

/**
 * Write a file
 * @async
 * @param {string} path - Virtual file path
 * @param {string} content - File content
 * @returns {Promise<Object>} File metadata
 */
async writeFile(path, content) {
    try {
        this.log.debug(`Writing file: ${path}`);

        // Check permission (more specific permission check could be done
        based on path)
        if
        (!this.kernel.modules.security.hasPermission('filesystem.write.*')) {
            throw new Error('Permission denied');
        }

        if (this.kernel.state.useLocalFilesystem) {
            // Call filesystem service
            const result = await
        this.kernel.services.get('filesystem').call('writeFile', {
            path,
            content
        });
            return result;
        } else {
            // Use localStorage-based filesystem
            const fs = this._getLocalStorage();

            // Get parent directory
            const parentDir = path.substring(0, path.lastIndexOf('/')) ||
        '/';
            const fileName = path.substring(path.lastIndexOf('/') + 1);

            // Make sure parent directory exists
            if (!fs[parentDir] || fs[parentDir].type !== 'directory') {
                throw new Error('Parent directory does not exist');
            }

            // Add file to parent if it doesn't exist
            if (!fs[path]) {
                if (!fs[parentDir].children.includes(fileName)) {
                    fs[parentDir].children.push(fileName);
                }
            }

            // Create or update file
        }
    }
}
```

```
        fs[path] = {
            type: 'file',
            content: content,
            created: fs[path] ? fs[path].created : Date.now(),
            modified: Date.now()
        };

        // Save filesystem
        this._saveLocalFilesystem(fs);

        return {
            path: path,
            size: content.length,
            modified: fs[path].modified
        };
    }
} catch (error) {
    this.log.error(`Failed to write file ${path}:`, error);
    throw error;
}
}

/**
 * Delete a file or directory
 * @async
 * @param {string} path - Virtual file path
 * @returns {Promise<boolean>} True if deleted successfully
 */
async deleteFile(path) {
    try {
        this.log.debug(`Deleting file/directory: ${path}`);

        // Check permission
        if (!this.kernel.modules.security.hasPermission('filesystem.delete.*')) {
            throw new Error('Permission denied');
        }

        if (this.kernel.state.useLocalFilesystem) {
            // Call filesystem service
            await
this.kernel.services.get('filesystem').call('deleteFile', { path });
            return true;
        } else {
            // Use localStorage-based filesystem
            const fs = this._getLocalFilesystem();

            if (!fs[path]) {
                throw new Error('File or directory not found');
            }

            // Get parent directory
            const parentDir = path.substring(0, path.lastIndexOf('/')) ||
'/';
        }
    }
}
```

```
const name = path.substring(path.lastIndexOf('/') + 1);

// Make sure parent directory exists
if (!fs[parentDir] || fs[parentDir].type !== 'directory') {
    throw new Error('Parent directory does not exist');
}

// If it's a directory, make sure it's empty
if (fs[path].type === 'directory' && fs[path].children.length
> 0) {
    throw new Error('Directory is not empty');
}

// Remove from parent
const childIndex = fs[parentDir].children.indexOf(name);
if (childIndex !== -1) {
    fs[parentDir].children.splice(childIndex, 1);
}

// Remove file or directory
delete fs[path];

// Save filesystem
this._saveLocalFilesystem(fs);

return true;
}
} catch (error) {
    this.log.error(`Failed to delete ${path}:`, error);
    throw error;
}
}

/**
 * List directory contents
 * @async
 * @param {string} path - Virtual directory path
 * @returns {Promise<Array>} Directory contents
 */
async listDirectory(path) {
    try {
        this.log.debug(`Listing directory: ${path}`);

        // Check permission
        if
(!this.kernel.modules.security.hasPermission('filesystem.read.*')) {
            throw new Error('Permission denied');
        }

        if (this.kernel.state.useLocalFilesystem) {
            // Call filesystem service
            const result = await
this.kernel.services.get('filesystem').call('listDirectory', { path });
            return result;
        }
    }
}
```

```
    } else {
        // Use localStorage-based filesystem
        const fs = this._getLocalFilesystem();

        if (!fs[path] || fs[path].type !== 'directory') {
            throw new Error('Directory not found');
        }

        const items = [];

        // Add items from directory
        for (const childName of fs[path].children) {
            const childPath = path === '/' ? `/ ${childName}` :
` ${path} / ${childName}`;
            const child = fs[childPath];

            if (!child) continue;

            const item = {
                name: childName,
                path: childPath,
                type: child.type,
                modified: child.modified
            };

            if (child.type === 'file') {
                item.size = child.content ? child.content.length : 0;
                item.extension = childName.includes('.') ?
                    childName.substring(childName.lastIndexOf('.') +
1) : '';
            }

            items.push(item);
        }

        // Sort directories first, then files
        items.sort((a, b) => {
            if (a.type === 'directory' && b.type !== 'directory')
return -1;
            if (a.type !== 'directory' && b.type === 'directory')
return 1;
            return a.name.localeCompare(b.name);
        });

        return items;
    }
} catch (error) {
    this.log.error(`Failed to list directory ${path}:`, error);
    throw error;
}
}

/**
 * Create a directory

```

```
* @async
* @param {string} path - Virtual directory path
* @returns {Promise<boolean>} True if created successfully
*/
async createDirectory(path) {
    try {
        this.log.debug(`Creating directory: ${path}`);

        // Check permission
        if (!this.kernel.modules.security.hasPermission('filesystem.write.*')) {
            throw new Error('Permission denied');
        }

        if (this.kernel.state.useLocalFilesystem) {
            // Call filesystem service
            await this.kernel.services.get('filesystem').call('createDirectory', { path });
            return true;
        } else {
            // Use localStorage-based filesystem
            const fs = this._getLocalFilesystem();

            // Check if directory already exists
            if (fs[path]) {
                throw new Error('Path already exists');
            }

            // Get parent directory
            const parentDir = path.substring(0, path.lastIndexOf('/')) || '/';
            const dirName = path.substring(path.lastIndexOf('/') + 1);

            // Make sure parent directory exists
            if (!fs[parentDir] || fs[parentDir].type !== 'directory') {
                throw new Error('Parent directory does not exist');
            }

            // Create directory
            fs[path] = {
                type: 'directory',
                children: [],
                created: Date.now(),
                modified: Date.now()
            };

            // Add to parent
            if (!fs[parentDir].children.includes(dirName)) {
                fs[parentDir].children.push(dirName);
            }

            // Save filesystem
            this._saveLocalFilesystem(fs);
        }
    } catch (error) {
        this.log.error(`Error creating directory ${path}: ${error.message}`);
    }
}
```

```
        return true;
    }
} catch (error) {
    this.log.error(`Failed to create directory ${path}:`, error);
    throw error;
}
}

/**
 * Get the local filesystem from localStorage
 * @private
 * @returns {Object} Filesystem object
 */
_getLocalFilesystem() {
    try {
        const fs = localStorage.getItem('gos_filesystem');
        return fs ? JSON.parse(fs) : {};
    } catch (error) {
        this.log.error('Failed to read filesystem from localStorage:', error);
        return {};
    }
}

/**
 * Save the local filesystem to localStorage
 * @private
 * @param {Object} fs - Filesystem object
 */
_saveLocalFilesystem(fs) {
    try {
        localStorage.setItem('gos_filesystem', JSON.stringify(fs));
    } catch (error) {
        this.log.error('Failed to save filesystem to localStorage:', error);
        throw error;
    }
}

/**
 * Process Manager
 * Manages application processes and windows
 */
class ProcessManager {
    /**
     * Create a new process manager
     * @param {Kernel} kernel - Kernel reference
     */
    constructor(kernel) {
        this.kernel = kernel;
        this.log = kernel._createLogger('process');
        this.processes = new Map();
        this.nextPid = 1000;
    }
}
```

```
        this.windowZIndex = 100;

        this.log.info('Process manager initialized');
    }

    /**
     * Create a new process for an application
     * @async
     * @param {string} appId - Application ID
     * @param {Object} appInfo - Application metadata
     * @param {Object} params - Launch parameters
     * @returns {Promise<Object>} Process object
    */
    async createProcess(appId, appInfo, params = {}) {
        try {
            const pid = this.nextPid++;

            this.log.info(`Creating process ${pid} for application ${appId}`);

            // Create window for the process
            const windowId = `window-${pid}`;
            const windowOptions = Object.assign({}, appInfo.window || {}, {
                help: appInfo.help || ''
            });
            const window = this._createWindow(windowId, appInfo.title || appId, appInfo.icon, windowOptions);

            // Create process object
            const process = {
                id: pid,
                appId,
                windowId,
                window,
                startTime: Date.now(),
                status: 'starting',
                params
            };

            // Store in process map
            this.processes.set(pid, process);

            // Initialize the application
            await this._initializeApplication(process, appInfo);

            // Update process status
            process.status = 'running';

            return process;
        } catch (error) {
            this.log.error(`Failed to create process for ${appId}:`, error);
            throw error;
        }
    }
}
```

```
* Terminate a process
* @async
* @param {number} pid - Process ID
* @returns {Promise<boolean>} True if terminated successfully
*/
async terminateProcess(pid) {
    try {
        if (!this.processes.has(pid)) {
            throw new Error(`Process not found: ${pid}`);
        }

        const process = this.processes.get(pid);
        this.log.info(`Terminating process ${pid} (${process.appId})`);

        // Update process status
        process.status = 'terminating';

        // Remove window
        this._removeWindow(process.windowId);

        // Remove taskbar item
        this._removeTaskbarItem(process.windowId);

        // Update process status and remove from list
        process.status = 'terminated';
        this.processes.delete(pid);

        // Emit terminated event
        this.kernel.modules.events.emit('application:terminated', {
            processId: pid,
            appId: process.appId
        });

        return true;
    } catch (error) {
        this.log.error(`Failed to terminate process ${pid}:`, error);
        throw error;
    }
}

/**
 * Get a list of all registered apps (system + installed)
 * @returns {Array} List of app information objects
 */
getRegisteredApps() {
    // Start with system apps
    let apps = [];

    // Try to get system apps from service
    try {
        this.kernel.services.get('apps').call('listApps')
            .then(systemApps => {
                apps = systemApps;
            })
    }
}
```

```
        .catch(err => {
            this.log.warn('Failed to get system apps', err);
        });
    } catch (e) {
        this.log.warn('Error accessing apps service', e);
    }

    // Add installed apps from localStorage
    try {
        const installedApps =
JSON.parse(localStorage.getItem('gos_installed_apps') || '[]');
        this.log.debug(`Found ${installedApps.length} installed apps`);

        installedApps.forEach(app => {
            if (app.desktopIcon !== false) {
                apps = [...apps, ...installedApps];
            }
        });
    } catch (e) {
        this.log.warn('Failed to load installed apps from localStorage',
e);
    }
}

return apps;
}

/**
 * Start an application by ID
 * @param {string} appId - Application ID to start
 * @param {Object} params - Launch parameters
 * @returns {Promise<string>} Process ID
 */
async startApp(appId, params = {}) {
    return this.kernel.launchApplication(appId, params);
}

/**
 * Create a window for an application
 * @private
 * @param {string} windowId - Window ID
 * @param {string} title - Window title
 * @param {string} icon - Window icon
 * @param {Object} options - Window options
 * @returns {HTMLElement} Window element
 */
_createWindow(windowId, title, icon, options = {}) {
    this.log.debug(`Creating window: ${windowId}`);

    const workspace = document.getElementById('workspace');

    // Create window element
    const window = document.createElement('div');
    window.id = windowId;
    window.className = 'window';
}
```

```
// Set initial position and size
const width = options.width || 800;
const height = options.height || 600;

// Center the window
const workspaceRect = workspace.getBoundingClientRect();
const left = Math.max(0, (workspaceRect.width - width) / 2);
const top = Math.max(0, (workspaceRect.height - height) / 2);

window.style.width = `${width}px`;
window.style.height = `${height}px`;
window.style.left = `${left}px`;
window.style.top = `${top}px`;
window.style.zIndex = this.windowZIndex++;

// Create window content
window.innerHTML =
    <div class="window-header">
        <div class="window-title">${title}</div>
        <div class="window-controls">
            <button class="window-control window-help"
title="Help">?</button>
            <button class="window-control window-minimize"
title="Minimize">_</button>
            <button class="window-control window-maximize"
title="Maximize">□</button>
            <button class="window-control window-close"
title="Close">×</button>
        </div>
    </div>
    <div class="window-content">
        <div class="app-loading">Loading ${title}...</div>
    </div>
`;

// Add window to workspace
workspace.appendChild(window);

// Add window to taskbar
this._addTaskBarItem(windowId, title, icon);

// Make window draggable
this._makeWindowDraggable(window);

// Make window resizable
if (options.resizable !== false) {
    this._makeWindowResizable(window);
}

// Set up window controls
const minimize = window.querySelector('.window-minimize');
const maximize = window.querySelector('.window-maximize');
const close = window.querySelector('.window-close');
```

```
const helpBtn = window.querySelector('.window-help');
const helpHtml = options.help || null;

minimize.addEventListener('click', () => {
    window.classList.add('minimized');

    // Update taskbar item
    const taskBarItem =
        document.getElementById(`taskbar-${windowId}`);
    if (taskBarItem) {
        taskBarItem.classList.remove('active');
    }
});

maximize.addEventListener('click', () => {
    window.classList.toggle('maximized');
});

close.addEventListener('click', () => {
    // Find the process by window ID and terminate it
    for (const [pid, process] of this.processes.entries()) {
        if (process.windowId === windowId) {
            this.terminateProcess(pid);
            break;
        }
    }
});

if (helpBtn && helpHtml !== null) {
    helpBtn.addEventListener('click', () => {
        this.kernel.modules.ui.showHelp(helpHtml || 'No help
available.');
    });
}

// Make window active when clicked
window.addEventListener('mousedown', () => {
    this._activateWindow(window);
});

return window;
}

/**
 * Remove a window
 * @private
 * @param {string} windowId - Window ID
 */
_removeWindow(windowId) {
    const window = document.getElementById(windowId);
    if (window) {
        window.remove();
    }
}
```

```
/**  
 * Add a taskbar item for a window  
 * @private  
 * @param {string} windowId - Window ID  
 * @param {string} title - Window title  
 * @param {string} icon - Window icon  
 */  
_addTaskbarItem(windowId, title, icon) {  
    const taskbarItems = document.getElementById('taskbar-items');  
  
    const item = document.createElement('div');  
    item.id = `taskbar-${windowId}`;  
    item.className = 'taskbar-item active';  
    item.innerHTML = `

>${icon} || ⌂</div>

>${title}</div>`;  
};  
  
// Toggle window visibility when taskbar item is clicked  
item.addEventListener('click', () => {  
    const window = document.getElementById(windowId);  
    if (window) {  
        if (window.classList.contains('minimized')) {  
            window.classList.remove('minimized');  
            item.classList.add('active');  
            this._activateWindow(window);  
        } else if (window === this._getActiveWindow()) {  
            window.classList.add('minimized');  
            item.classList.remove('active');  
        } else {  
            this._activateWindow(window);  
        }  
    }  
});  
  
taskbarItems.appendChild(item);  
}  
  
/**  
 * Remove a taskbar item  
 * @private  
 * @param {string} windowId - Window ID  
 */  
_removeTaskbarItem(windowId) {  
    const taskbarItem = document.getElementById(`taskbar-${windowId}`);  
    if (taskbarItem) {  
        taskbarItem.remove();  
    }  
}  
  
/**  
 * Make a window draggable  
 * @private  
 */


```

```
* @param {HTMLElement} window - Window element
*/
_makeWindowDraggable(window) {
    const header = window.querySelector('.window-header');

    let isDragging = false;
    let offsetX = 0;
    let offsetY = 0;

    header.addEventListener('mousedown', (e) => {
        // Ignore if clicking on window controls
        if (e.target.closest('.window-controls')) {
            return;
        }

        // Activate window
        this._activateWindow(window);

        // Don't drag if maximized
        if (window.classList.contains('maximized')) {
            return;
        }

        isDragging = true;

        const rect = window.getBoundingClientRect();
        offsetX = e.clientX - rect.left;
        offsetY = e.clientY - rect.top;

        // Set initial cursor
        document.body.style.cursor = 'move';

        // Prevent text selection while dragging
        e.preventDefault();
    });

    document.addEventListener('mousemove', (e) => {
        if (!isDragging) return;

        const workspaceRect =
document.getElementById('workspace').getBoundingClientRect();

        // Calculate new position
        let left = e.clientX - offsetX;
        let top = e.clientY - offsetY;

        // Keep window within workspace bounds
        left = Math.max(0, Math.min(left, workspaceRect.width - 100));
        top = Math.max(0, Math.min(top, workspaceRect.height - 30));

        window.style.left = `${left}px`;
        window.style.top = `${top}px`;
    });
}
```

```
document.addEventListener('mouseup', () => {
    if (isDragging) {
        isDragging = false;
        document.body.style.cursor = '';
    }
});

/***
 * Make a window resizable
 * @private
 * @param {HTMLElement} window - Window element
 */
_makeWindowResizable(window) {
    // Minimum dimensions
    const minWidth = 300;
    const minHeight = 200;

    // Create resize handle
    const resizeHandle = document.createElement('div');
    resizeHandle.style.position = 'absolute';
    resizeHandle.style.right = '0';
    resizeHandle.style.bottom = '0';
    resizeHandle.style.width = '15px';
    resizeHandle.style.height = '15px';
    resizeHandle.style.cursor = 'nwse-resize';
    resizeHandle.style.zIndex = '10';

    window.appendChild(resizeHandle);

    let isResizing = false;
    let startX = 0;
    let startY = 0;
    let startWidth = 0;
    let startHeight = 0;

    resizeHandle.addEventListener('mousedown', (e) => {
        // Don't resize if maximized
        if (window.classList.contains('maximized')) {
            return;
        }

        isResizing = true;

        startX = e.clientX;
        startY = e.clientY;
        startWidth = window.offsetWidth;
        startHeight = window.offsetHeight;

        // Set cursor
        document.body.style.cursor = 'nwse-resize';

        // Prevent text selection while resizing
        e.preventDefault();
    });
}
```

```
// Activate window
this._activateWindow(window);
});

document.addEventListener('mousemove', (e) => {
    if (!isResizing) return;

    // Calculate new dimensions
    const width = Math.max(minWidth, startWidth + (e.clientX -
startX));
    const height = Math.max(minHeight, startHeight + (e.clientY -
startY));

    window.style.width = `${width}px`;
    window.style.height = `${height}px`;
});

document.addEventListener('mouseup', () => {
    if (isResizing) {
        isResizing = false;
        document.body.style.cursor = '';
    }
});
}

/***
 * Activate a window (bring to front)
 * @private
 * @param {HTMLElement} window - Window element
 */
_activateWindow(window) {
    // Update z-index
    window.style.zIndex = this.windowZIndex++;

    // Update active status in taskbar
    const windowId = window.id;
    const taskbarItems = document.querySelectorAll('.taskbar-item');

    taskbarItems.forEach(item => {
        item.classList.remove('active');
    });

    const activeItem = document.getElementById(`taskbar-${windowId}`);
    if (activeItem) {
        activeItem.classList.add('active');
    }
}

/***
 * Get the currently active window
 * @private
 * @returns {HTMLElement|null} Active window element
 */
```

```
_getActiveWindow() {
    const windows = document.querySelectorAll('.window');
    let activeWindow = null;
    let highestZIndex = -1;

    windows.forEach(window => {
        const zIndex = parseInt(window.style.zIndex || 0);
        if (zIndex > highestZIndex) {
            highestZIndex = zIndex;
            activeWindow = window;
        }
    });
}

return activeWindow;
}

/**
 * Initialize an application in a window
 * @private
 * @async
 * @param {Object} process - Process object
 * @param {Object} appInfo - Application metadata
 */
async _initializeApplication(process, appInfo) {
    try {
        const windowContent = document.querySelector(`#${process.windowId}
>window-content`);

        // Clear loading indicator
        windowContent.innerHTML = '';

        // Create app container
        const appContainer = document.createElement('div');
        appContainer.className = `app-${appInfo.id}`;
        appContainer.style.height = '100%';
        windowContent.appendChild(appContainer);

        // Initialize app based on appId
        switch (appInfo.id) {
            case 'mathematicalSandbox':
                this._initializeMathematicalSandbox(appContainer, process,
appInfo);
                break;

            case 'fileManager':
                this._initializeFileManager(appContainer, process,
appInfo);
                break;

            case 'terminal':
                this._initializeTerminal(appContainer, process, appInfo);
                break;

            case 'editor':
                break;
        }
    } catch (error) {
        console.error(`Error initializing application ${appInfo.id}: ${error}`);
    }
}
```

```
        this._initializeCodeEditor(appContainer, process,
appInfo);
        break;

    case 'settings':
        this._initializeSettings(appContainer, process, appInfo);
        break;

    case 'appStore':
        this._initializeAppStore(appContainer, process,
appInfo);
        break;

    case 'diagnostics':
        this._initializeDiagnostics(appContainer, process,
appInfo);
        break;

    case 'dev-center': // <-- Corrected ID
        this._initializeDeveloperCenter(appContainer, process,
appInfo);
        break;

    default:
        appContainer.innerHTML =
            `<div style="padding: 20px; text-align: center;">
                <h3>Application ${appInfo.id} not implemented</h3>
                <p>This application is not available in the
current version.</p>
            </div>
`;
    }

    this.log.info(`Application ${appInfo.id} initialized in window
${process.windowId}`);
} catch (error) {
    this.log.error(`Failed to initialize application ${appInfo.id}:`, error);

    // Show error message in window
    const windowContent = document.querySelector(`#${process.windowId}
>window-content`);
    windowContent.innerHTML =
        `<div style="padding: 20px; color: #f44336;">
            <h3>Application Error</h3>
            <p>${error.message}</p>
            <pre style="margin-top: 10px; background: var(--terminal-
bg); padding: 10px; overflow: auto; font-size: 12px;">${error.stack || ''}</pre>
        </div>
`;

    throw error;
}
}
```

```
// This method is inside the ProcessManager class in index.php

_initializeDeveloperCenter(container, process, options) {
    container.innerHTML =
        <div style="padding: 20px; font-family: sans-serif;">
            <h2>Developer Submission Portal</h2>
            <p>Submit your application for review. Please provide a valid
JSON manifest and your application's JavaScript code.</p>
            <div id="dev-form-status" style="margin-bottom: 15px; padding:
10px; border-radius: 3px; display: none;"></div>
            <form id="dev-submit-form">
                <label for="manifest" style="display: block; margin-
bottom: 5px;"><strong>Manifest (JSON)</strong></label>
                <textarea id="manifest" name="manifest" rows="10" required
style="width: 100%; padding: 8px; border: 1px solid var(--main-border); border-
radius: 3px; font-family: monospace;"></textarea>
                <br><br>
                <label for="code" style="display: block; margin-bottom:
5px;"><strong>Application Code (JavaScript)</strong></label>
                <textarea id="code" name="code" rows="15" required
style="width: 100%; padding: 8px; border: 1px solid var(--main-border); border-
radius: 3px; font-family: monospace;"></textarea>
                <br><br>
                <button type="submit" class="button button-primary">Submit
for Review</button>
            </form>
        </div>
    `;

    const form = container.querySelector('#dev-submit-form');
    const statusDiv = container.querySelector('#dev-form-status');

    form.addEventListener('submit', (e) => {
        e.preventDefault();
        statusDiv.style.display = 'none';

        const manifestText = container.querySelector('#manifest').value;
        const codeText = container.querySelector('#code').value;

        // --- Frontend Validation ---
        let manifest;
        try {
            manifest = JSON.parse(manifestText);
        } catch (err) {
            statusDiv.textContent = 'Error: Manifest is not valid JSON.';
            statusDiv.style.backgroundColor = '#f8d7da';
            statusDiv.style.color = '#721c24';
            statusDiv.style.display = 'block';
            return;
        }

        // ✅ FIXED: Validation now correctly checks for 'title' and
        'entry' to match the backend.
    });
}
```



```
class="button">Redo</button>
            <button id="${process.windowId}-math-menu">
class="button">Math</button>
            <button id="${process.windowId}-help-btn">
class="button">Help</button>
        </div>

        <div id="${process.windowId}-setup-panel" style="padding: 20px;">
            <h3>Grid Configuration</h3>
            <div style="margin-bottom: 10px;">
                <label>Cell Size (pixels): </label>
                <input type="number" id="${process.windowId}-cell-size" min="50" value="70">
            </div>
            <div style="margin-bottom: 10px;">
                <label>Grid Rows: </label>
                <input type="number" id="${process.windowId}-grid-rows" min="1" value="4">
            </div>
            <div style="margin-bottom: 10px;">
                <label>Grid Columns: </label>
                <input type="number" id="${process.windowId}-grid-cols" min="1" value="4">
            </div>
            <button id="${process.windowId}-create-grid" class="button button-primary">Create Grid</button>
        </div>

        <div id="${process.windowId}-grid-container" class="grid-container" style="display: none;">
            <div id="${process.windowId}-grid" class="grid"></div>
        </div>

        <div id="${process.windowId}-status-bar" style="padding: 5px; border-top: 1px solid var(--main-border); font-size: 12px;">
            Ready.
        </div>
    </div>

    <!-- Cell Properties Modal -->
    <div id="${process.windowId}-cell-modal" style="display: none; position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); background-color: var(--window-body); border: 1px solid var(--main-border); padding: 15px; z-index: 1000; width: 300px;">
        <h3>Cell Properties</h3>
        <div style="margin-bottom: 10px;">
            <label>Text: </label>
            <input type="text" id="${process.windowId}-cell-text">
        </div>
        <div style="margin-bottom: 10px;">
            <label>Expression: </label>
            <input type="text" id="${process.windowId}-cell-expression">
        </div>
    </div>
```

```
</div>
<div style="margin-bottom: 10px;">
    <label>Background Color: </label>
    <input type="color" id="${process.windowId}-cell-bg-color"
value="#222222">
</div>
<div style="margin-bottom: 10px;">
    <label>Font Size: </label>
    <input type="range" id="${process.windowId}-cell-font-size" min="6" max="72" value="12">
        <span id="${process.windowId}-font-size-value">12</span>
</div>
<div style="margin-bottom: 10px;">
    <label>Font Color: </label>
    <input type="color" id="${process.windowId}-cell-font-color"
value="#33ff33">
</div>
<div style="text-align: right;">
    <button id="${process.windowId}-apply-cell" class="button
button-primary">Apply</button>
    <button id="${process.windowId}-cancel-cell" class="button">Cancel</button>
</div>
`;

// App state
const state = {
    squareData: {},           // Cell data
    splitSquares: {},         // Split cells info
    actionHistory: [],         // For undo
    redoStack: [],             // For redo
    squareSize: 70,            // Default cell size
    numRows: 4,                // Default rows
    numCols: 4,                // Default columns
    currentCell: null          // Currently selected cell
};

// Initialize event handlers for grid creation
document.getElementById(`#${process.windowId}-create-
grid`).addEventListener('click', () => {
    createGrid();
});

document.getElementById(`#${process.windowId}-new-
grid`).addEventListener('click', () => {
    if (confirm('Create a new grid? All unsaved changes will be
lost.')) {
        document.getElementById(`#${process.windowId}-setup-
panel`).style.display = 'block';
        document.getElementById(`#${process.windowId}-grid-
container`).style.display = 'none';
        state.squareData = {};
        state.splitSquares = {};
    }
});
```

```
        state.actionHistory = [];
        state.redoStack = [];
    }
});

document.getElementById(`#${process.windowId}-save-grid`).addEventListener('click', () => {
    saveGrid();
});

document.getElementById(`#${process.windowId}-load-grid`).addEventListener('click', () => {
    loadGrid();
});

document.getElementById(`#${process.windowId}-undo-btn`).addEventListener('click', () => {
    undo();
});

document.getElementById(`#${process.windowId}-redo-btn`).addEventListener('click', () => {
    redo();
});

document.getElementById(`#${process.windowId}-help-btn`).addEventListener('click', () => {
    alert('Mathematical Sandbox Help\n\n' +
        'Left Click: Edit cell\n' +
        'Right Click: Split/merge cell\n' +
        'Expressions: Use standard JavaScript math expressions
(e.g., sin(x), 2+3)\n' +
        'Math Menu: Access additional mathematical tools');
});

document.getElementById(`#${process.windowId}-cancel-cell`).addEventListener('click', () => {
    document.getElementById(`#${process.windowId}-cell-modal`).style.display = 'none';
});

document.getElementById(`#${process.windowId}-apply-cell`).addEventListener('click', () => {
    applyCellProperties();
});

document.getElementById(`#${process.windowId}-cell-font-size`).addEventListener('input', function() {
    document.getElementById(`#${process.windowId}-font-size-value`).textContent = this.value;
});

// Function to create the grid
function createGrid() {
```

```
        state.squareSize =
parseInt(document.getElementById(`#${process.windowId}-cell-size`).value) || 70;
        state numRows =
parseInt(document.getElementById(`#${process.windowId}-grid-rows`).value) || 4;
        state numCols =
parseInt(document.getElementById(`#${process.windowId}-grid-cols`).value) || 4;

        // Hide setup, show grid
        document.getElementById(`#${process.windowId}-setup-
panel`).style.display = 'none';
        document.getElementById(`#${process.windowId}-grid-
container`).style.display = 'block';

        // Create grid
        drawGrid();

        // Update status
        updateStatus('Grid created');
    }

// Function to draw the grid
function drawGrid() {
    const grid = document.getElementById(`#${process.windowId}-grid`);
    grid.innerHTML = '';

    // Set grid template
    grid.style.gridTemplateColumns = `repeat(${state.numCols},
${state.squareSize}px)`;

    // Create cells
    for (let i = 0; i < state.numRows; i++) {
        for (let j = 0; j < state.numCols; j++) {
            const cell = document.createElement('div');
            cell.className = 'cell';
            cell.dataset.row = i;
            cell.dataset.col = j;

            // Add event listeners
            cell.addEventListener('click', handleCellClick);
            cell.addEventListener('contextmenu',
handleCellRightClick);

            grid.appendChild(cell);
        }
    }
}

// Handle cell click
function handleCellClick(e) {
    const cell = e.currentTarget;
    const row = parseInt(cell.dataset.row);
    const col = parseInt(cell.dataset.col);
    const subRow = cell.dataset.subrow !== undefined ?
parseInt(cell.dataset.subrow) : undefined;
```

```
        const subCol = cell.dataset.subcol !== undefined ?
parseInt(cell.dataset.subcol) : undefined;

            openCellProperties(row, col, subRow, subCol);
        }

// Handle cell right click
function handleCellRightClick(e) {
    e.preventDefault();

    const cell = e.currentTarget;
    const row = parseInt(cell.dataset.row);
    const col = parseInt(cell.dataset.col);

    const key = `${row}_${col}`;

    if (state.splitSquares[key]) {
        // If already split, merge
        if (confirm('Merge this cell?')) {
            mergeCell(row, col);
        }
    } else {
        // If not split, offer to split
        const splitSize = prompt('Enter split size (rows,cols):',
'2,2');
        if (splitSize) {
            const [rows, cols] = splitSize.split(',').map(Number);
            if (rows > 0 && cols > 0) {
                splitCell(row, col, rows, cols);
            }
        }
    }
}

// Open cell properties
function openCellProperties(row, col, subRow, subCol) {
    const key = subRow !== undefined && subCol !== undefined ?
`${row}_${col}_${subRow}_${subCol}` : `${row}_${col}`;

    const cellData = state.squareData[key] || {};
    // Populate form
    document.getElementById(`#${process.windowId}-cell-text`).value =
cellData.text || '';
    document.getElementById(`#${process.windowId}-cell-
expression`).value = cellData.expression || '';
    document.getElementById(`#${process.windowId}-cell-bg-color`).value =
cellData.color || '#222222';
    document.getElementById(`#${process.windowId}-cell-font-
size`).value = cellData.fontSize || 12;
    document.getElementById(`#${process.windowId}-font-size-
value`).textContent = cellData.fontSize || 12;
    document.getElementById(`#${process.windowId}-cell-font-
color`).value = cellData.fontColor || '#33ff33';
}
```

```
// Store current cell
state.currentCell = { row, col, subRow, subCol };

// Show modal
document.getElementById(`#${process.windowId}-cell-modal`).style.display = 'block';
}

// Apply cell properties
function applyCellProperties() {
    if (!state.currentCell) return;

    const { row, col, subRow, subCol } = state.currentCell;
    const key = subRow !== undefined && subCol !== undefined ?
        `${row}_${col}_${subRow}_${subCol}` : `${row}_${col}`;

    // Get form values
    const text = document.getElementById(`#${process.windowId}-cell-text`).value;
    const expression = document.getElementById(`#${process.windowId}-cell-expression`).value;
    const color = document.getElementById(`#${process.windowId}-cell-bg-color`).value;
    const fontSize =
parseInt(document.getElementById(`#${process.windowId}-cell-font-size`).value);
    const fontColor = document.getElementById(`#${process.windowId}-cell-font-color`).value;

    // Save previous state for undo
    const prevState = state.squareData[key] ? {
...state.squareData[key] } : null;
    state.actionHistory.push({ key, prevState });
    state.redoStack = []; // Clear redo stack

    // Update data
    state.squareData[key] = {
        text,
        expression,
        color,
        fontSize,
        fontColor
    };

    // Update display
    updateCellDisplay(row, col, subRow, subCol);

    // Hide modal
    document.getElementById(`#${process.windowId}-cell-modal`).style.display = 'none';

    // Update status
    updateStatus('Cell updated');
}
```

```
// Update cell display
function updateCellDisplay(row, col, subRow, subCol) {
    let cell;
    const key = subRow !== undefined && subCol !== undefined ?
        `${row}_${col}_${subRow}_${subCol}` : `${row}_${col}`;

    if (subRow !== undefined && subCol !== undefined) {
        // Find sub-cell
        const mainCell = document.querySelector(`#${process.windowId}-grid .cell[data-row="${row}"][data-col="${col}"]`);
        cell = mainCell.querySelector(`.sub-cell[data-subrow="${subRow}"][data-subcol="${subCol}"]`);
    } else {
        // Find main cell
        cell = document.querySelector(`#${process.windowId}-grid .cell[data-row="${row}"][data-col="${col}"]`);
    }

    if (!cell) return;

    const cellData = state.squareData[key] || {};

    // Apply styling
    cell.style.backgroundColor = cellData.color || '#222222';
    cell.style.color = cellData.textColor || '#33ff33';
    cell.style.fontSize = `${cellData.fontSize || 12}px`;

    // Set content
    let displayText = cellData.text || '';

    // Evaluate expression
    if (cellData.expression) {
        try {
            // Basic expression evaluation
            const expr = cellData.expression
                .replace(/sin\(/g, 'Math.sin(')
                .replace(/cos\(/g, 'Math.cos(')
                .replace(/tan\(/g, 'Math.tan(')
                .replace(/sqrt\(/g, 'Math.sqrt(')
                .replace(/abs\(/g, 'Math.abs(')
                .replace(/pi/g, 'Math.PI')
                .replace(/\^/g, '**');

            const result = eval(expr);
            displayText = String(result);
        } catch (e) {
            displayText = "Error";
        }
    }

    cell.textContent = displayText;
}
```

```
// Split a cell
function splitCell(row, col, rows, cols) {
    const key = `${row}_${col}`;

    // Save state for undo
    state.actionHistory.push({
        type: 'split',
        key,
        prevState: {
            isSplit: false,
            data: state.squareData[key]
        }
    });
    state.redoStack = [];

    // Mark as split
    state.splitSquares[key] = { rows, cols };

    // Get the cell
    const cell = document.querySelector(`#${process.windowId}-grid
.cell[data-row="${row}"][data-col="${col}"]`);
    cell.classList.add('split');
    cell.innerHTML = '';
    cell.style.gridTemplateColumns = `repeat(${cols}, 1fr)`;

    // Create sub-cells
    for (let i = 0; i < rows; i++) {
        for (let j = 0; j < cols; j++) {
            const subCell = document.createElement('div');
            subCell.className = 'sub-cell';
            subCell.dataset.row = row;
            subCell.dataset.col = col;
            subCell.dataset.subrow = i;
            subCell.dataset.subcol = j;

            // Add event listeners
            subCell.addEventListener('click', handleCellClick);

            cell.appendChild(subCell);
        }
    }

    updateStatus('Cell split');
}

// Merge a split cell
function mergeCell(row, col) {
    const key = `${row}_${col}`;

    if (!state.splitSquares[key]) return;

    // Save state for undo
    const subCells = {};
    const subCellElements =
```

```
document.querySelectorAll(`#${process.windowId}-grid .cell[data-row="${row}"] [data-col="${col}"] .sub-cell`);

    subCellElements.forEach(subCell => {
        const subRow = parseInt(subCell.dataset.subrow);
        const subCol = parseInt(subCell.dataset.subcol);
        const subKey = `${row}_${col}_${subRow}_${subCol}`;
        subCells[subKey] = state.squareData[subKey];
    });

    state.actionHistory.push({
        type: 'merge',
        key,
        prevState: {
            isSplit: true,
            splitInfo: { ...state.splitSquares[key] },
            subCells: subCells
        }
    });
    state.redoStack = [];

    // Remove split flag
    delete state.splitSquares[key];

    // Remove sub-cell data
    Object.keys(state.squareData).forEach(dataKey => {
        if (dataKey.startsWith(`${key}_`)) {
            delete state.squareData[dataKey];
        }
    });

    // Reset cell
    const cell = document.querySelector(`#${process.windowId}-grid
.cell[data-row="${row}"][data-col="${col}"]`);
    cell.classList.remove('split');
    cell.innerHTML = '';
    cell.style.removeProperty('grid-template-columns');

    // Update display
    updateCellDisplay(row, col);

    updateStatus('Cell merged');
}

// Undo function
function undo() {
    if (state.actionHistory.length === 0) {
        alert('Nothing to undo');
        return;
    }

    const action = state.actionHistory.pop();

    if (action.type === 'split') {
```

```
// Undo split (merge the cell)
const [row, col] = action.key.split('_').map(Number);

// Save current state for redo
const subCells = {};
const subCellElements =
document.querySelectorAll(`#${process.windowId}-grid .cell[data-row="${row}"] [data-col="${col}"] .sub-cell`);

subCellElements.forEach(subCell => {
    const subRow = parseInt(subCell.dataset.subrow);
    const subCol = parseInt(subCell.dataset.subcol);
    const subKey = `${row}_${col}_${subRow}_${subCol}`;
    subCells[subKey] = state.squareData[subKey];
});

state.redoStack.push({
    type: 'split',
    key: action.key,
    redoState: {
        isSplit: true,
        splitInfo: { ...state.splitSquares[action.key] },
        subCells: subCells
    }
});

// Merge the cell
mergeCell(row, col);

// Restore previous cell state
if (action.prevState.data) {
    state.squareData[action.key] = action.prevState.data;
    updateCellDisplay(row, col);
}
} else if (action.type === 'merge') {
    // Undo merge (re-split the cell)
    const [row, col] = action.key.split('_').map(Number);

    // Save current state for redo
    state.redoStack.push({
        type: 'merge',
        key: action.key,
        redoState: {
            isSplit: false,
            data: state.squareData[action.key]
        }
    });

    // Restore the split
    const splitInfo = action.prevState.splitInfo;
    splitCell(row, col, splitInfo.rows, splitInfo.cols);

    // Restore sub-cell data
    for (let subKey in action.prevState.subCells) {
```

```
        if (action.prevState.subCells[subKey]) {
            state.squareData[subKey] =
            action.prevState.subCells[subKey];

                // Extract sub-row and sub-col from key
                const [_, __, subRow, subCol] =
            subKey.split('_').map(Number);
                updateCellDisplay(row, col, subRow, subCol);
            }
        }
    } else {
        // Regular cell update
        const key = action.key;
        let row, col, subRow, subCol;

        if (key.split('_').length === 4) {
            [row, col, subRow, subCol] = key.split('_').map(Number);
        } else {
            [row, col] = key.split('_').map(Number);
        }

        // Save current state for redo
        state.redoStack.push({
            key: key,
            redoState: { ...state.squareData[key] }
        });

        // Restore previous state
        if (action.prevState) {
            state.squareData[key] = action.prevState;
        } else {
            delete state.squareData[key];
        }

        // Update display
        updateCellDisplay(row, col, subRow, subCol);
    }

    updateStatus('Undo completed');
}

// Redo function
function redo() {
    if (state.redoStack.length === 0) {
        alert('Nothing to redo');
        return;
    }

    const action = state.redoStack.pop();

    if (action.type === 'split') {
        // Redo split
        const [row, col] = action.key.split('_').map(Number);
    }
}
```

```
// Save current state for undo
state.actionHistory.push({
    type: 'split',
    key: action.key,
    prevState: {
        isSplit: false,
        data: state.squareData[action.key]
    }
});

// Redo the split
const splitInfo = action.redoState.splitInfo;
splitCell(row, col, splitInfo.rows, splitInfo.cols);

// Restore sub-cell data
for (let subKey in action.redoState.subCells) {
    if (action.redoState.subCells[subKey]) {
        state.squareData[subKey] =
action.redoState.subCells[subKey];

            // Extract sub-row and sub-col from key
            const [_, __, subRow, subCol] =
subKey.split('_').map(Number);
            updateCellDisplay(row, col, subRow, subCol);
        }
    }
} else if (action.type === 'merge') {
    // Redo merge
    const [row, col] = action.key.split('_').map(Number);

    // Save current state for undo
    const subCells = {};
    const subCellElements =
document.querySelectorAll(`#${process.windowId}-grid .cell[data-row="${row}"]
[data-col="${col}"] .sub-cell`);

    subCellElements.forEach(subCell => {
        const subRow = parseInt(subCell.dataset.subrow);
        const subCol = parseInt(subCell.dataset.subcol);
        const subKey = `${row}_${col}_${subRow}_${subCol}`;
        subCells[subKey] = state.squareData[subKey];
    });
}

state.actionHistory.push({
    type: 'merge',
    key: action.key,
    prevState: {
        isSplit: true,
        splitInfo: { ...state.splitSquares[action.key] },
        subCells: subCells
    }
});

// Merge the cell
```

```
        mergeCell(row, col);

        // Restore cell state
        if (action.redoState.data) {
            state.squareData[action.key] = action.redoState.data;
            updateCellDisplay(row, col);
        }
    } else {
        // Regular cell update
        const key = action.key;
        let row, col, subRow, subCol;

        if (key.split('_').length === 4) {
            [row, col, subRow, subCol] = key.split('_').map(Number);
        } else {
            [row, col] = key.split('_').map(Number);
        }

        // Save current state for undo
        state.actionHistory.push({
            key: key,
            prevState: state.squareData[key] ? {
                ...state.squareData[key]
            } : null
        });

        // Restore redo state
        state.squareData[key] = action.redoState;

        // Update display
        updateCellDisplay(row, col, subRow, subCol);
    }

    updateStatus('Redo completed');
}

// Save grid to file
function saveGrid() {
    // Create a data object with all grid info
    const gridData = {
        squareData: state.squareData,
        splitSquares: state.splitSquares,
        numRows: state.numRows,
        numCols: state.numCols,
        squareSize: state.squareSize
    };

    // Ask for filename
    const fileName = prompt("Enter filename to save:", "grid.json");
    if (!fileName) return;

    const username = this.kernel?.currentUser?.username || 'guest';
    const path = `/users/${username}/Documents/${fileName}`;

    // Try to save via filesystem
}
```

```
        try {
            this.kernel.modules.filesystem.writeFile(path,
JSON.stringify(gridData, null, 2))
            .then(() => {
                updateStatus(`Grid saved to ${path}`);
            })
            .catch(error => {
                // Fallback to localStorage
                localStorage.setItem(`grid_${fileName}`,
JSON.stringify(gridData));
                updateStatus(`Grid saved to local storage:
${fileName}`);
            });
        } catch (error) {
            // Fallback to localStorage
            localStorage.setItem(`grid_${fileName}`,
JSON.stringify(gridData));
            updateStatus(`Grid saved to local storage: ${fileName}`);
        }
    }

    // Load grid from file
    function loadGrid() {
        // Ask for filename
        const fileName = prompt("Enter filename to load:", "grid.json");
        if (!fileName) return;

        const username = this.kernel?.currentUser?.username || 'guest';
        const path = `/users/${username}/Documents/${fileName}`;

        // Try to load via filesystem
        try {
            this.kernel.modules.filesystem.readFile(path)
            .then(result => {
                const gridData = JSON.parse(result.content);
                loadGridData(gridData);
            })
            .catch(error => {
                // Try localStorage
                const savedGrid =
localStorage.getItem(`grid_${fileName}`);
                if (savedGrid) {
                    const gridData = JSON.parse(savedGrid);
                    loadGridData(gridData);
                    updateStatus(`Grid loaded from local storage:
${fileName}`);
                } else {
                    alert(`Error loading grid: File not found`);
                }
            });
        } catch (error) {
            // Try localStorage
            const savedGrid = localStorage.getItem(`grid_${fileName}`);
            if (savedGrid) {
```

```
        const gridData = JSON.parse(savedGrid);
        loadGridData(gridData);
        updateStatus(`Grid loaded from local storage:
${fileName}`);
    } else {
        alert(`Error loading grid: ${error.message}`);
    }
}

// Load grid data
function loadGridData(gridData) {
    // Validate data structure
    if (!gridData.squareData || !gridData numRows ||
!gridData.numCols) {
        alert('Invalid grid data format');
        return;
    }

    // Load the data
    state.squareData = gridData.squareData;
    state.splitSquares = gridData.splitSquares || {};
    state.numRows = gridData numRows;
    state.numCols = gridData.numCols;
    state.squareSize = gridData.squareSize;

    // Clear history
    state.actionHistory = [];
    state.redoStack = [];

    // Switch to grid view
    document.getElementById(`#${process.windowId}-setup-
panel`).style.display = 'none';
    document.getElementById(`#${process.windowId}-grid-
container`).style.display = 'block';

    // Draw the grid
    drawGrid();

    // Update cell displays
    for (let key in state.squareData) {
        let row, col, subRow, subCol;

        if (key.split('_').length === 4) {
            [row, col, subRow, subCol] = key.split('_').map(Number);

            // Make sure parent cell is split
            const parentKey = `${row}_${col}`;
            if (state.splitSquares[parentKey]) {
                const cell =
document.querySelector(`#${process.windowId}-grid .cell[data-row="${row}"][data-
col="${col}"]`);

                if (!cell.classList.contains('split')) {
                    const splitInfo = state.splitSquares[parentKey];

```

```
        splitCell(row, col, splitInfo.rows,
splitInfo.cols);
    }
}

updateCellDisplay(row, col, subRow, subCol);
} else {
    [row, col] = key.split('_').map(Number);
    updateCellDisplay(row, col);
}
}

updateStatus(`Grid loaded successfully`);
}

// Update status bar
function updateStatus(message) {
    document.getElementById(`#${process.windowId}-status-
bar`).textContent = message;
}

/***
 * Initialize File Manager application
 * @private
 * @param {HTMLElement} container - Container element
 * @param {Object} process - Process object
 * @param {Object} appInfo - Application metadata
 */
_initializeFileManager(container, process, appInfo) {
    // Create file manager UI
    container.innerHTML =
        <div class="file-browser">
            <div class="file-toolbar">
                <button id="${process.windowId}-back" class="button"
title="Back"></button>
                <button id="${process.windowId}-forward" class="button"
title="Forward">></button>
                <button id="${process.windowId}-up" class="button"
title="Up">↑</button>
                <button id="${process.windowId}-refresh" class="button"
title="Refresh">⟳</button>
                <span style="margin: 0 10px;">Path:</span>
                <input type="text" id="${process.windowId}-path"
style="flex: 1; min-width: 200px;">
                <button id="${process.windowId}-go" class="button">Go</button>
            </div>
            <div class="file-browser-content">
                <div class="file-sidebar">
                    <div class="file-sidebar-item" data-
path="/users/${this.kernel.currentUser.username}">
                        <div class="file-sidebar-icon">👤 </div>
                        <div>My Files</div>
                    </div>
                </div>
            </div>
        </div>
}
```

```
</div>
<div class="file-sidebar-item" data-path="/users/${this.kernel.currentUser.username}/Desktop">
    <div class="file-sidebar-icon">💻</div>
    <div>Desktop</div>
</div>
<div class="file-sidebar-item" data-path="/users/${this.kernel.currentUser.username}/Documents">
    <div class="file-sidebar-icon">📄</div>
    <div>Documents</div>
</div>
<div class="file-sidebar-item" data-path="/apps">
    <div class="file-sidebar-icon">📦</div>
    <div>Applications</div>
</div>
<div class="file-sidebar-item" data-path="/system">
    <div class="file-sidebar-icon">⚙️</div>
    <div>System</div>
</div>
<div id="${process.windowId}-file-main" class="file-main">
    <div id="${process.windowId}-file-list" class="file-list"></div>
</div>
</div>
</div>
`;
```

// File manager state

```
const state = {
    currentPath: `/users/${this.kernel.currentUser.username}`,
    selectedFile: null,
    history: [`/users/${this.kernel.currentUser.username}`],
    historyIndex: 0
};
```

// Initialize path input

```
document.getElementById(`#${process.windowId}-path`).value =
state.currentPath;
```

// Load current directory

```
this._loadDirectory(process.windowId, state.currentPath, state);
```

// Set active sidebar item

```
const initialSidebarItem = container.querySelector(`.file-sidebar-item[data-path="${state.currentPath}"]`);
if (initialSidebarItem) {
    initialSidebarItem.classList.add('active');
}
```

// Set up event handlers

```
document.getElementById(`#${process.windowId}-back`).addEventListener('click', () => {
    if (state.historyIndex > 0) {
```

```
        state.historyIndex--;
        const path = state.history[state.historyIndex];
        this._loadDirectory(process.windowId, path, state);
        document.getElementById(`#${process.windowId}-path`).value =
    path;
    }
});

document.getElementById(`#${process.windowId}-
forward`).addEventListener('click', () => {
    if (state.historyIndex < state.history.length - 1) {
        state.historyIndex++;
        const path = state.history[state.historyIndex];
        this._loadDirectory(process.windowId, path, state);
        document.getElementById(`#${process.windowId}-path`).value =
    path;
    }
});

document.getElementById(`#${process.windowId}-
up`).addEventListener('click', () => {
    const parentPath = state.currentPath.substring(0,
state.currentPath.lastIndexOf('/')) || '/';
    this._navigateTo(process.windowId, parentPath, state);
});

document.getElementById(`#${process.windowId}-
refresh`).addEventListener('click', () => {
    this._loadDirectory(process.windowId, state.currentPath, state);
});

document.getElementById(`#${process.windowId}-
go`).addEventListener('click', () => {
    const path = document.getElementById(`#${process.windowId}-
path`).value;
    this._navigateTo(process.windowId, path, state);
});

document.getElementById(`#${process.windowId}-
path`).addEventListener('keypress', (e) => {
    if (e.key === 'Enter') {
        const path = e.target.value;
        this._navigateTo(process.windowId, path, state);
    }
});

// Side panel navigation
const sidebar = container.querySelector('.file-sidebar');
sidebar.addEventListener('click', (e) => {
    const item = e.target.closest('.file-sidebar-item');
    if (item) {
        const path = item.getAttribute('data-path');
        this._navigateTo(process.windowId, path, state);
    }
});
```

```
// Update active item
sidebar.querySelectorAll('.file-sidebar-item').forEach(el => {
    el.classList.remove('active');
});
item.classList.add('active');
}

});

/***
 * Load directory contents in file manager
 * @private
 * @param {string} windowId - Window ID
 * @param {string} path - Directory path
 * @param {Object} state - File manager state
 */
_loadDirectory(windowId, path, state) {
    const fileList = document.getElementById(`#${windowId}-file-list`);
    fileList.innerHTML = '<div style="text-align: center; padding: 20px;">Loading...</div>';

    // Update state
    state.currentPath = path;

    // List directory contents
    this.kernel.modules.filesystem.listDirectory(path).then(items => {
        // Clear file list
        fileList.innerHTML = '';

        if (items.length === 0) {
            fileList.innerHTML = '<div style="text-align: center; padding: 20px; opacity: 0.7;">This folder is empty</div>';
            return;
        }

        // Add each item to the file list
        items.forEach(item => {
            const itemElement = document.createElement('div');
            itemElement.className = 'file-item';
            itemElement.dataset.path = item.path;
            itemElement.dataset.type = item.type;

            const icon = item.type === 'directory' ? '📁' : this._getFileIcon(item.extension || '');

            itemElement.innerHTML = `
                <div class="file-icon">${icon}</div>
                <div class="file-name">${item.name}</div>
            `;

            // Add click handler
            itemElement.addEventListener('click', (e) => {
                // Deselect any previously selected item
                fileList.querySelectorAll('.file-item').forEach(el => {
```

```
        el.classList.remove('selected');
    });

    // Select this item
    itemElement.classList.add('selected');
    state.selectedFile = item.path;
});

// Add double-click handler
itemElement.addEventListener('dblclick', (e) => {
    if (item.type === 'directory') {
        // Navigate to directory
        this._navigateTo(windowId, item.path, state);
    } else {
        // Open file
        this._openFile(item.path);
    }
});

fileList.appendChild(itemElement);
});

}).catch(error => {
    fileList.innerHTML =
        <div style="text-align: center; padding: 20px; color:
#f44336;">
            Error loading directory: ${error.message}
        </div>
    ;
});

/** 
 * Navigate to a directory in file manager
 * @private
 * @param {string} windowId - Window ID
 * @param {string} path - Directory path
 * @param {Object} state - File manager state
 */
_navigateTo(windowId, path, state) {

    // --- SECURITY CHECK (JAIL) ---
    const user = this.kernel.currentUser;
    const isAdmin = user && user.roles.includes('admin');
    //const isDeveloper = user && user.roles.includes('developer');

    //if (!isAdmin && !isDeveloper) {
    if (!isAdmin) {
        const userHomeDir = `/users/${user.username}`;
        // Ensure the requested path is within the user's own home
        directory.
        if (!path.startsWith(userHomeDir)) {
            this.kernel.modules.ui.showNotification('Access Denied', 'You
can only browse your user directory.', 3000);
            return; // Abort the navigation.
    }
}
```

```
        }
    }
    // --- END SECURITY CHECK ---

    // Update path input
    document.getElementById(`#${windowId}-path`).value = path;

    // Add to history if it's a new path
    if (path !== state.currentPath) {
        if (state.historyIndex < state.history.length - 1) {
            state.history = state.history.slice(0, state.historyIndex +
1);
        }

        state.history.push(path);
        state.historyIndex = state.history.length - 1;
    }

    // Load directory
    this._loadDirectory(windowId, path, state);
}

/**
 * Open a file
 * @private
 * @param {string} path - File path
 */
_openFile(path) {
    // Determine file type
    const extension = path.split('.').pop().toLowerCase();

    // Launch appropriate app based on file type
    if (['txt', 'md', 'js', 'json', 'html', 'css',
'php'].includes(extension)) {
        this.kernel.launchApplication('editor', { filePath: path });
    } else {
        // For other file types
        this.kernel.modules.ui.showNotification(
            'File',
            `Opening ${path.split('/').pop()}`,
            2000
        );
    }

    // You could add handlers for other file types here
}
}

/**
 * Get an icon for a file extension
 * @private
 * @param {string} extension - File extension
 * @returns {string} Icon emoji
 */
_getFileIcon(extension) {
```

```
const icons = {
    'txt': '📝',
    'md': '📝',
    'js': '📄',
    'json': '📋',
    'html': '🌐',
    'css': 'styleType',
    'php': '_PHP_',
    'jpg': '🖼',
    'jpeg': '🖼',
    'png': '🖼',
    'gif': '🖼',
    'pdf': '📄',
    'zip': '🗄',
    'mp3': '🎵',
    'mp4': '🎥'
};

return icons[extension.toLowerCase()] || '📄';
}

/**
 * Initialize Terminal application
 * @private
 * @param {HTMLElement} container - Container element
 * @param {Object} process - Process object
 * @param {Object} appInfo - Application metadata
 */
_initializeTerminal(container, process, appInfo) {
    // Create terminal UI
    container.innerHTML =
        `

<div id="${process.windowId}-output" class="terminal-output">
</div>
            <div class="terminal-input-line">
                <div id="${process.windowId}-prompt" class="terminal-prompt">guest@gos:~$</div>
                    <input type="text" id="${process.windowId}-input" class="terminal-input" autofocus>
                </div>
            </div>
        `;

    // Terminal state
    const state = {
        history: [],
        historyIndex: -1,
        currentDirectory: `/users/${this.kernel.currentUser.username}`
    };

    // Initial output
    const outputElement = document.getElementById(`#${process.windowId}-output`);
    outputElement.innerHTML = `


```

```
<div style="color: var(--highlight);">Text Reversion Terminal
v1.0</div>
<div>Type 'help' for a list of available commands.</div>
<div>&nbsp;</div>
`;

// Update prompt
this._updateTerminalPrompt(process.windowId, state);

// Handle input
const inputElement = document.getElementById(`#${process.windowId}-
input`);
inputElement.addEventListener('keydown', (e) => {
    if (e.key === 'Enter') {
        const command = inputElement.value;

        if (command.trim()) {
            // Add to history
            state.history.push(command);
            state.historyIndex = state.history.length;

            // Show command in output
            const promptText =
document.getElementById(`#${process.windowId}-prompt`).textContent;
            this._appendTerminalOutput(process.windowId, `<span
style="color: var(--highlight);">${promptText}</span> ${command}`);
        }

        // Execute command
        this._executeTerminalCommand(process.windowId, command,
state);

        // Clear input
        inputElement.value = '';
    }
} else if (e.key === 'ArrowUp') {
    // Navigate history up
    if (state.history.length > 0 && state.historyIndex > 0) {
        state.historyIndex--;
        inputElement.value = state.history[state.historyIndex];
    }
    e.preventDefault();
} else if (e.key === 'ArrowDown') {
    // Navigate history down
    if (state.historyIndex < state.history.length - 1) {
        state.historyIndex++;
        inputElement.value = state.history[state.historyIndex];
    } else {
        // Clear input at end of history
        state.historyIndex = state.history.length;
        inputElement.value = '';
    }
    e.preventDefault();
}
});
```

```
// Focus input when terminal clicked
container.addEventListener('click', () => {
    inputElement.focus();
});

/**
 * Update terminal prompt
 * @private
 * @param {string} windowId - Window ID
 * @param {Object} state - Terminal state
 */
_updateTerminalPrompt(windowId, state) {
    const promptElement = document.getElementById(`#${windowId}-prompt`);
    const username = this.kernel.currentUser.username;

    // Format the directory for display
    let displayPath = state.currentDirectory;
    if (displayPath.startsWith(`/users/${username}`)) {
        displayPath = '~' +
displayPath.substring(`/users/${username}`.length) || '~';
    }

    promptElement.textContent = `${username}@gos:${displayPath}`;
}

/**
 * Append output to terminal
 * @private
 * @param {string} windowId - Window ID
 * @param {string} text - Output text
 */
_appendTerminalOutput(windowId, text) {
    const outputElement = document.getElementById(`#${windowId}-output`);
    outputElement.innerHTML += `<div>${text}</div>`;
    outputElement.scrollTop = outputElement.scrollHeight;
}

/**
 * Execute a terminal command
 * @private
 * @param {string} windowId - Window ID
 * @param {string} command - Command text
 * @param {Object} state - Terminal state
 */
_executeTerminalCommand(windowId, command, state) {
    // Parse command and arguments
    const parts = command.split(' ');
    const cmd = parts[0].toLowerCase();
    const args = parts.slice(1);

    // Execute command
    switch (cmd) {
```

```
        case 'help':
            this._appendTerminalOutput(windowId, `

                Available commands:
                <br>help - Show this help
                <br>ls, dir [path] - List directory contents
                <br>cd [path] - Change directory
                <br>pwd - Print working directory
                <br>cat, type [file] - Show file content
                <br>mkdir [path] - Create directory
                <br>touch [file] [content] - Create file
                <br>rm, del [file] - Remove file
                <br>clear, cls - Clear terminal
                <br>echo [text] - Display text
                <br>whoami - Show current user
                <br>date - Show current date
            `);
            break;

        case 'ls':
        case 'dir':
            this._terminalListDirectory(windowId, args[0] ||
state.currentDirectory, state);
            break;

        case 'cd':
            this._terminalChangeDirectory(windowId, args[0], state);
            break;

        case 'pwd':
            this._appendTerminalOutput(windowId, state.currentDirectory);
            break;

        case 'cat':
        case 'type':
            if (!args[0]) {
                this._appendTerminalOutput(windowId, 'Usage: cat [file]');
            } else {
                this._terminalShowFile(windowId, args[0], state);
            }
            break;

        case 'mkdir':
            if (!args[0]) {
                this._appendTerminalOutput(windowId, 'Usage: mkdir
[directory]');
            } else {
                this._terminalCreateDirectory(windowId, args[0], state);
            }
            break;

        case 'touch':
        case 'new':
            if (!args[0]) {
                this._appendTerminalOutput(windowId, 'Usage: touch [file]
```

```
[content']);
        } else {
            const content = args.slice(1).join(' ');
            this._terminalCreateFile(windowId, args[0], content,
state);
        }
        break;

    case 'rm':
    case 'del':
        if (!args[0]) {
            this._appendTerminalOutput(windowId, 'Usage: rm [file]');
        } else {
            this._terminalRemoveFile(windowId, args[0], state);
        }
        break;

    case 'clear':
    case 'cls':
        document.getElementById(`#${windowId}-output`).innerHTML = '';
        break;

    case 'echo':
        this._appendTerminalOutput(windowId, args.join(' '));
        break;

    case 'whoami':
        this._appendTerminalOutput(windowId,
this.kernel.currentUser.username);
        break;

    case 'date':
        this._appendTerminalOutput(windowId, new Date().toString());
        break;

    default:
        this._appendTerminalOutput(windowId, `Command not found:
${cmd}. Type 'help' for available commands.`);
    }
}

/***
 * List directory contents in terminal
 * @private
 * @param {string} windowId - Window ID
 * @param {string} path - Directory path
 * @param {Object} state - Terminal state
 */
_terminalListDirectory(windowId, path, state) {
    // Resolve relative path
    path = this._resolveTerminalPath(path, state.currentDirectory);

    this.kernel.modules.filesystem.listDirectory(path).then(items => {
        if (items.length === 0) {
```

```
        this._appendTerminalOutput(windowId, `Directory is empty:  
${path}`);  
        return;  
    }  
  
    let output = '';  
  
    // Add directories  
    items.filter(item => item.type === 'directory')  
        .forEach(item => {  
            output += ``>${item.name}/</span> `;  
        });  
  
    // Add files  
    items.filter(item => item.type !== 'directory')  
        .forEach(item => {  
            output += `${item.name} `;  
        });  
  
    this._appendTerminalOutput(windowId, output);  
}).catch(error => {  
    this._appendTerminalOutput(windowId, `Error: ${error.message}`);  
});  
}  
  
/**  
 * Normalizes a client-side path, resolving '..' and '.' segments.  
 * @private  
 * @param {string} path - The path to normalize.  
 * @returns {string} The normalized, absolute path.  
 */  
_normalizePath(path) {  
    const parts = path.split('/').filter(p => p.length > 0);  
    const absolutes = [];  
    for (const part of parts) {  
        if (part === '.') {  
            continue;  
        }  
        if (part === '..') {  
            if (absolutes.length > 0) {  
                absolutes.pop();  
            }  
        } else {  
            absolutes.push(part);  
        }  
    }  
    // Return the joined path, ensuring it starts with a single '/'.  
    return '/' + absolutes.join('/');  
}  
  
/**  
 * Change directory in terminal, with security jail for non-admin users.  
 * @private  
 */
```

```
* @param {string} windowId - Window ID
* @param {string} path - Directory path
* @param {Object} state - Terminal state
*/
_terminalChangeDirectory(windowId, path, state) {
    if (!path) {
        state.currentDirectory =
`/users/${this.kernel.currentUser.username}`;
        this._updateTerminalPrompt(windowId, state);
        return;
    }

    // Resolve the path parts first, then normalize to handle '..'
    const resolvedPath = this._resolveTerminalPath(path,
state.currentDirectory);
    const normalizedPath = this._normalizePath(resolvedPath);

    // --- SECURITY CHECK (JAIL) ---
    const user = this.kernel.currentUser;
    const isAdmin = user && user.roles.includes('admin');
    //const isDeveloper = user && user.roles.includes('developer');

    //if (!isAdmin && !isDeveloper) {
    if (!isAdmin) {
        const userHomeDir = `/users/${user.username}`;
        // Check the final, normalized path against the user's home
        directory.
        if (!normalizedPath.startsWith(userHomeDir)) {
            this._appendTerminalOutput(windowId, `Error: Permission
denied. Cannot navigate outside your home directory.`);
            return; // Abort the directory change.
        }
    }
    // --- END SECURITY CHECK ---

    // Verify the normalized directory exists and update the state.
    this.kernel.api.filesystem.call('listDirectory', { path:
normalizedPath }).then(() => {
        state.currentDirectory = normalizedPath;
        this._updateTerminalPrompt(windowId, state);
    }).catch(error => {
        this._appendTerminalOutput(windowId, `Error: ${error.message}`);
    });
}

/**
 * Show file content in terminal
 * @private
 * @param {string} windowId - Window ID
 * @param {string} path - File path
 * @param {Object} state - Terminal state
*/
_terminalShowFile(windowId, path, state) {
```

```
// Resolve path
const targetPath = this._resolveTerminalPath(path,
state.currentDirectory);

    this.kernel.modules.filesystem.readFile(targetPath).then(result => {
        this._appendTerminalOutput(windowId, result.content);
    }).catch(error => {
        this._appendTerminalOutput(windowId, `Error: ${error.message}`);
    });
}

/**
 * Create directory in terminal
 * @private
 * @param {string} windowId - Window ID
 * @param {string} path - Directory path
 * @param {Object} state - Terminal state
 */
_terminalCreateDirectory(windowId, path, state) {
    // Resolve path
    const targetPath = this._resolveTerminalPath(path,
state.currentDirectory);

    this.kernel.modules.filesystem.createDirectory(targetPath).then(() =>
{
    this._appendTerminalOutput(windowId, `Directory created:
${targetPath}`);
}).catch(error => {
    this._appendTerminalOutput(windowId, `Error: ${error.message}`);
});
}

/**
 * Create file in terminal
 * @private
 * @param {string} windowId - Window ID
 * @param {string} path - File path
 * @param {string} content - File content
 * @param {Object} state - Terminal state
 */
_terminalCreateFile(windowId, path, content, state) {
    // Resolve path
    const targetPath = this._resolveTerminalPath(path,
state.currentDirectory);

    this.kernel.modules.filesystem.writeFile(targetPath, content).then(() => {
        this._appendTerminalOutput(windowId, `File created:
${targetPath}`);
    }).catch(error => {
        this._appendTerminalOutput(windowId, `Error: ${error.message}`);
    });
}
```

```
/**  
 * Remove file in terminal  
 * @private  
 * @param {string} windowId - Window ID  
 * @param {string} path - File path  
 * @param {Object} state - Terminal state  
 */  
_terminalRemoveFile(windowId, path, state) {  
    // Resolve path  
    const targetPath = this._resolveTerminalPath(path,  
state.currentDirectory);  
  
    this.kernel.modules.filesystem.deleteFile(targetPath).then(() => {  
        this._appendTerminalOutput(windowId, `Removed: ${targetPath}`);  
    }).catch(error => {  
        this._appendTerminalOutput(windowId, `Error: ${error.message}`);  
    });  
}  
  
/**  
 * Resolve a path in terminal  
 * @private  
 * @param {string} path - Path to resolve  
 * @param {string} currentDir - Current directory  
 * @returns {string} Resolved path  
 */  
_resolveTerminalPath(path, currentDir) {  
    if (!path) return currentDir;  
  
    // Handle home directory  
    if (path === '~') {  
        return `/users/${this.kernel.currentUser.username}`;  
    }  
  
    if (path.startsWith('~/')) {  
        return  
`/users/${this.kernel.currentUser.username}${path.substring(1)}`;  
    }  
  
    // Absolute path  
    if (path.startsWith('/')) {  
        return path;  
    }  
  
    // Relative path  
    return currentDir === '/' ? `${path}` : `${currentDir}/${path}`;  
}  
  
_initializeCodeEditor(container, process, appInfo) {  
    container.innerHTML = `<div class="editor"><div class="editor-toolbar"><button id="${process.windowId}-new" class="button">New</button>
```

```
        <button id="${process.windowId}-open" class="button">Open</button>
        <button id="${process.windowId}-save" class="button button-primary">Save</button>
        <span id="${process.windowId}-filename" style="margin-left: 20px; opacity: 0.7;">Untitled</span>
    </div>
    <div class="editor-content">
        <textarea id="${process.windowId}-editor" class="editor-textarea" spellcheck="false"></textarea>
    </div>
</div>
`;

const state = { currentFile: null, modified: false };
const editorTextarea = document.getElementById(`#${process.windowId}-editor`);
const filenameSpan = document.getElementById(`#${process.windowId}-filename`);

// --- SECURITY HELPER FUNCTION ---
// This function checks if a given path is within the user's home directory.
const isPathAllowed = (path) => {
    if (!path) return false;

    const user = this.kernel.currentUser;
    const isAdmin = user && user.roles.includes('admin');

    // Admins are always allowed full access.
    if (isAdmin) {
        return true;
    }

    // For standard users, the path must be within their home directory.
    const normalizedPath = this._normalizePath(path);
    const userHomeDir = `/users/${user.username}`;
    return normalizedPath.startsWith(userHomeDir);
};

// --- END SECURITY HELPER ---

//  NEW: Apply tab size from UI settings
editorTextarea.style.tabSize = this.kernel.modules.ui.currentTabSize || 4;

// Initialize event handlers
document.getElementById(`#${process.windowId}-new`).addEventListener('click', () => {
    if (state.modified && !confirm('You have unsaved changes. Discard them?')) {
        return;
    }
})
```

```
        document.getElementById(`#${process.windowId}-editor`).value = '';
        document.getElementById(`#${process.windowId}-
filename`).textContent = 'Untitled';
        state.currentFile = null;
        state.modified = false;
    });

    // SECURED "OPEN" ACTION
    document.getElementById(`#${process.windowId}-
open`).addEventListener('click', () => {
    if (state.modified && !confirm('You have unsaved changes. Discard them?')) {
        return;
    }
    const path = prompt('Enter file path to open:',
`/users/${this.kernel.currentUser.username}/`);
    if (!path) return;

    // --- SECURITY CHECK ---
    if (!isPathAllowed(path)) {
        this.kernel.modules.ui.showNotification('Access Denied', 'You can only open files from your home directory.', 4000);
        return;
    }
    // --- END CHECK ---

    this.kernel.modules.filesystem.readFile(path).then(result => {
        editorTextarea.value = result.content;
        filenameSpan.textContent = path;
        state.currentFile = path;
        state.modified = false;
    }).catch(error => {
        this.kernel.modules.ui.showNotification('Error', `Could not open file: ${error.message}`, 5000);
    });
});

// SECURED "SAVE" ACTION
document.getElementById(`#${process.windowId}-
save`).addEventListener('click', () => {
    const content = editorTextarea.value;

    if (state.currentFile) {
        // A simple save on an already opened/allowed file does not need a new check.
        this.kernel.modules.filesystem.writeFile(state.currentFile,
content).then(() => {
            this.kernel.modules.ui.showNotification('Saved', `File saved: ${state.currentFile}`, 2000);
            state.modified = false;
            filenameSpan.textContent = state.currentFile;
        }).catch(error => {
            this.kernel.modules.ui.showNotification('Error', `Could not save file: ${error.message}`, 5000);
        });
    }
});
```

```
        });
    } else {
        // "Save As..." prompts for a new path.
        const path = prompt('Enter file path to save:',
`/users/${this.kernel.currentUser.username}/untitled.txt`);
        if (!path) return;

        // --- SECURITY CHECK ---
        if (!isPathAllowed(path)) {
            this.kernel.modules.ui.showNotification('Access Denied',
'You can only save files within your home directory.', 4000);
            return;
        }
        // --- END CHECK ---

        this.kernel.modules.filesystem.writeFile(path,
content).then(() => {
        filenameSpan.textContent = path;
        state.currentFile = path;
        state.modified = false;
        this.kernel.modules.ui.showNotification('Saved', `File
saved: ${path}`, 2000);
    }).catch(error => {
        this.kernel.modules.ui.showNotification('Error', `Could
not save file: ${error.message}`, 5000);
    });
})
});

//  NEW: Enhanced keydown handler for Tab and Shift+Tab support
editorTextarea.addEventListener('keydown', (e) => {
    if (e.key === 'Tab') {
        e.preventDefault();
        const tabSize = this.kernel.modules.ui.currentTabSize || 4;
        const tab = ' '.repeat(tabSize);
        const start = editorTextarea.selectionStart;
        const end = editorTextarea.selectionEnd;

        if (e.shiftKey) { // Handle outdenting (Shift+Tab)
            const lineStart = editorTextarea.value.lastIndexOf('\n',
start - 1) + 1;
            if (editorTextarea.value.substring(lineStart, lineStart +
tabSize) === tab) {
                editorTextarea.setRangeText('', lineStart, lineStart +
tabSize, 'end');
            }
        } else { // Handle indenting (Tab)
            editorTextarea.setRangeText(tab, start, end, 'end');
        }
        state.modified = true;
        filenameSpan.textContent = state.currentFile ?
`${state.currentFile} *` : 'Untitled *';
    }
});
```

```
// Track modifications
document.getElementById(`#${process.windowId}-editor`).addEventListener('input', () => {
    if (!state.modified) {
        state.modified = true;
        const filename = document.getElementById(`#${process.windowId}-filename`);
        filename.textContent = `${filename.textContent}`;
    }
});

// SECURED INITIAL FILE LOAD FROM LAUNCH PARAMETERS
if (process.params.filePath) {
    // --- SECURITY CHECK ---
    if (!isPathAllowed(process.params.filePath)) {
        this.kernel.modules.ui.showNotification('Access Denied',
'Cannot open file outside your home directory.', 4000);
        filenameSpan.textContent = 'Access Denied';
        editorTextarea.value = '## ACCESS DENIED ##\n\nYou do not have
permission to open this file.';
        editorTextarea.disabled = true;
        return;
    }
    // --- END CHECK ---

this.kernel.modules.filesystem.readFile(process.params.filePath).then(result => {
    editorTextarea.value = result.content;
    filenameSpan.textContent = process.params.filePath;
    state.currentFile = process.params.filePath;
    state.modified = false;
}).catch(error => {
    this.kernel.modules.ui.showNotification('Error', `Could not
open file: ${error.message}`, 5000);
});
}

/**
 * Initialize Settings application
 * @private
 */
_initializeSettings(container, process, appInfo) {
    // MODIFIED: Add "Account" and "User Management" to the sidebar.
    let adminNav = '';
    if (this.kernel.modules.security.hasPermission('users.manage')) {
        adminNav = `<div class="settings-nav-item" data-section="user-
management">User Management</div>`;
    }

    container.innerHTML =
        <div class="settings">
```

```
        <div class="settings-sidebar">
            <div class="settings-nav-item active" data-
section="account">My Account</div>
            <div class="settings-nav-item" data-
section="appearance">Appearance</div>
                ${adminNav}
            <div class="settings-nav-item" data-
section="about">About</div>
        </div>
        <div id="${process.windowId}-settings-content"
class="settings-content"></div>
    </div>
    `;

    this._loadSettingsSection(process.windowId, 'account');

    const sidebar = container.querySelector('.settings-sidebar');
    sidebar.addEventListener('click', (e) => {
        const item = e.target.closest('.settings-nav-item');
        if (item) {
            sidebar.querySelectorAll('.settings-nav-item').forEach(el =>
el.classList.remove('active'));
            item.classList.add('active');
            const section = item.getAttribute('data-section');
            this._loadSettingsSection(process.windowId, section);
        }
    });
}

/**
 * Load a settings section
 * @private
 * @param {string} windowId - Window ID
 * @param {string} section - Section name
 */
_loadSettingsSection(windowId, section) {
    const content = document.getElementById(`${windowId}-settings-
content`);

    switch (section) {
        // NEW: "My Account" section for changing password.
        case 'account':
            const currentUser = this.kernel.currentUser;
            content.innerHTML =
                <div class="settings-section">
                    <h2 class="settings-section-title">My Account</h2>
                    <p>Username: ${currentUser.username}</p>
                    <p>Name: ${currentUser.name}</p>
                    <p>Roles: ${currentUser.roles.join(', ')})</p>
                </div>
                <div class="settings-section">
                    <h3 class="settings-section-title">Change
Password</h3>
                    <div class="form-group">
```

```
        <label for="current-password">Current  
Password</label>  
        <input type="password" id="current-password">  
    </div>  
    <div class="form-group">  
        <label for="new-password">New Password</label>  
        <input type="password" id="new-password">  
    </div>  
    <button id="change-password-btn" class="button button-primary">Update Password</button>  
    </div>  
    `;  
    document.getElementById('change-password-btn').onclick = async  
(() => {  
    const currentUser = await this.kernel.modules.auth.getCurrentUser();  
    const currentPassword = document.getElementById('current-  
password').value;  
    const newPassword = document.getElementById('new-  
password').value;  
    try {  
        await this.kernel.api.users.setPassword({  
username: currentUser.username, currentPassword, newPassword });  
        this.kernel.modules.ui.showNotification('Success',  
'Password updated successfully.');  
        document.getElementById('current-password').value = '';  
        document.getElementById('new-password').value = '';  
    } catch(e) {  
        this.kernel.modules.ui.showNotification('Error',  
e.message, 5000);  
    }  
};  
break;  
  
// NEW: "User Management" section for admins.  
case 'user-management':  
    content.innerHTML = `<h2 class="settings-section-title">User  
Management</h2><div id="user-list">Loading...</div>`;  
    this._renderUserManagement(windowId);  
    break;  
  
case 'appearance':  
    content.innerHTML = `<div class="settings-section">  
        <h2 class="settings-section-title">Appearance  
Settings</h2>  
  
        <div class="settings-option">  
            <label class="settings-label">Theme</label>  
            <select id="${windowId}-theme" class="settings-  
select">  
                <option value="vintage">Vintage</option>  
                <option value="dark">Dark</option>  
                <option value="light">Light</option>  
                <option value="blue">Blue</option>
```

```
        </select>
    </div>

    <div class="settings-option">
        <label class="settings-label">Font Size</label>
        <select id="${windowId}-font-size"
class="settings-select">
            <option value="small">Small</option>
            <option value="medium">Medium</option>
            <option value="large">Large</option>
        </select>
    </div>

    <div class="settings-option">
        <label class="settings-label">
            <input type="checkbox" id="${windowId}-
animations" class="settings-checkbox">
                Enable animations
        </label>
    </div>

    <div class="settings-option">
        <label class="settings-label">
            <input type="checkbox" id="${windowId}-crt-
effects" class="settings-checkbox">
                Enable CRT effects
        </label>
    </div>

    <div class="settings-option">
        <button id="${windowId}-save-appearance"
class="button button-primary">Save Changes</button>
        <button id="${windowId}-reset-appearance"
class="button">Reset to Defaults</button>
    </div>
</div>
`;

// Set current values
document.getElementById(`#${windowId}-theme`).value =
this.kernel.modules.ui.currentTheme || 'vintage';
document.getElementById(`#${windowId}-font-size`).value =
this.kernel.modules.ui.currentFontSize || 'medium';
document.getElementById(`#${windowId}-animations`).checked =
this.kernel.modules.ui.animations !== false;
document.getElementById(`#${windowId}-crt-effects`).checked =
this.kernel.modules.ui.crtEffects === true;

// Save button handler
document.getElementById(`#${windowId}-save-
appearance`).addEventListener('click', () => {
    const theme = document.getElementById(`#${windowId}-
theme`).value;
    const fontSize = document.getElementById(`#${windowId}-
```

```
font-size`).value;
    const animations = document.getElementById(`#${windowId}-animations`).checked;
    const crtEffects = document.getElementById(`#${windowId}-crt-effects`).checked;

    // Apply changes
    this.kernel.modules.ui.applyTheme(theme);
    this.kernel.modules.ui.applyFontSize(fontSize);
    this.kernel.modules.ui.setAnimations(animations);
    this.kernel.modules.ui.setCrtEffects(crtEffects);

    // Show notification
    this.kernel.modules.ui.showNotification('Settings',
    'Appearance settings saved', 2000);
}

// Reset button handler
document.getElementById(`#${windowId}-reset-appearance`).addEventListener('click', () => {
    document.getElementById(`#${windowId}-theme`).value =
    'vintage';
    document.getElementById(`#${windowId}-font-size`).value =
    'medium';
    document.getElementById(`#${windowId}-animations`).checked =
    true;
    document.getElementById(`#${windowId}-crt-effects`).checked =
    false;
});
break;

case 'system':
    content.innerHTML = `
        <div class="settings-section">
            <h2 class="settings-section-title">System
    Settings</h2>

            <div class="settings-option">
                <label class="settings-label">
                    <input type="checkbox" id="${windowId}-debug"
class="settings-checkbox">
                        Enable debug mode
                </label>
            </div>

            <div class="settings-option">
                <button id="${windowId}-save-system" class="button
button-primary">Save Changes</button>
            </div>

            <div class="settings-section" style="margin-top:
    30px;">
                <h3 class="settings-section-title">System
    Maintenance</h3>

```

```
        <div class="settings-option">
            <button id="${windowId}-clear-storage"
class="button">Clear Local Storage</button>
        </div>

        <div class="settings-option">
            <button id="${windowId}-restart"
class="button">Restart System</button>
        </div>
    </div>
`;

// Set current values
document.getElementById(`#${windowId}-debug`).checked =
this.kernel.state.debug;

// Save button handler
document.getElementById(`#${windowId}-save-
system`).addEventListener('click', () => {
    const debug = document.getElementById(`#${windowId}-
debug`).checked;

    // Apply changes
    this.kernel.state.debug = debug;

    // Show notification
    this.kernel.modules.ui.showNotification('Settings',
'System settings saved', 2000);
});

// Clear storage button handler
document.getElementById(`#${windowId}-clear-
storage`).addEventListener('click', () => {
    if (confirm('This will clear all stored data. Are you
sure?')) {
        localStorage.clear();
        this.kernel.modules.ui.showNotification('System',
'Local storage cleared', 2000);
    }
});

// Restart button handler
document.getElementById(`#${windowId}-
restart`).addEventListener('click', () => {
    if (confirm('Are you sure you want to restart the
system?')) {
        window.location.reload();
    }
});
break;

case 'about':
```

```
const systemInfo = this.kernel.getSystemInfo();

content.innerHTML = `
    <div class="settings-section">
        <h2 class="settings-section-title">About
            Text Reversion</h2>

            <div style="text-align: center; margin-
                bottom: 20px;">
                <div style="font-size: 72px;
                    margin-bottom: 10px;">G</div>
                <h3>Genesis Operating System</h3>
                <div>Version ${systemInfo.version}
                    (${systemInfo.buildDate})</div>
            </div>

            <div class="settings-option">
                <label class="settings-
                    label">System Information</label>
                <div style="background-color:
                    var(--terminal-bg); padding: 15px; border-radius: 4px;">
                    <div>Version:
                        ${systemInfo.version}</div>
                    <div>Build Date:
                        ${systemInfo.buildDate}</div>
                    <div>Code Name:
                        ${systemInfo.codename}</div>
                    <div>Uptime:
                        ${Math.floor(systemInfo.uptime / 60)} minutes, ${systemInfo.uptime % 60} seconds</div>
                    <div>User:
                        ${systemInfo.user ? systemInfo.user.username : 'Not logged in'}</div>
                    <div>Filesystem:
                        ${systemInfo.useLocalFilesystem ? 'Server-based' : 'LocalStorage-based'}</div>
                </div>
            </div>

            <div class="settings-option"
                style="margin-top: 20px;">
                <label class="settings-
                    label">Credits</label>
                <div style="background-color:
                    var(--terminal-bg); padding: 10px; border-radius: 4px; font-size: 13px;">
                    Text Reversion is an open-
                    source experiment demonstrating a monolithic browser OS.<br>
                    Created by the Text
                    Reversion Team.
                </div>
            </div>

            <div class="settings-option"
                style="margin-top: 20px;">
                <label class="settings-
                    label">Change Log</label>
```

```
                                <div style="background-color:  
var(--terminal-bg); padding: 10px; border-radius: 4px; font-size: 13px; max-  
height:150px; overflow:auto;">  
                                <ul>  
                                    <li>v1.0.0 -  
Initial release</li>  
                                    <li>v1.0.1 - Minor  
fixes and improvements</li>  
                                </ul>  
                            </div>  
  
                            <div class="settings-option"  
style="margin-top: 20px;">  
                            <div style="text-align: center;  
opacity: 0.7;">  
© 2025 Text Reversion  
Team<br>  
                            All rights reserved.  
                            </div>  
                        </div>  
                    </div>  
                `;  
            break;  
  
        default:  
            content.innerHTML = `  
                <div class="settings-section">  
                    <h2 class="settings-section-  
title">${section.charAt(0).toUpperCase() + section.slice(1)} Settings</h2>  
                    <p>This section is not yet implemented.</p>  
                </div>  
            `;  
        }  
    }  
  
    /**  
     * Initializes the App Store application UI.  
     * This updated version includes a "Submissions" tab for administrators.  
     * @param {HTMLElement} container - The content container of the app  
window.  
     * @param {object} process - The process object from the kernel.  
     * @param {object} options - Launch options.  
     */  
    _initializeAppStore(container, process, options) {  
        // --- 1. Basic UI Structure & Styles ---  
        container.innerHTML = `  
            <style>  
                .appstore-container { display: flex; flex-direction: column;  
height: 100%; background-color: var(--window-bg); color: var(--text-color); }  
                .appstore-nav { display: flex; border-bottom: 1px solid var(--  
border-color); padding: 0 10px; }  
                .appstore-tab { padding: 10px 15px; cursor: pointer; border-  
bottom: 2px solid transparent; }  
            </style>
```

```
.appstore-tab.active { border-bottom-color: var(--accent-color); font-weight: bold; }
.appstore-content { flex-grow: 1; overflow-y: auto; }
.appstore-pane { display: none; padding: 20px; }
.appstore-pane.active { display: block; }
.app-grid, .submission-list { display: grid; gap: 20px; }
.app-grid { grid-template-columns: repeat(auto-fill, minmax(150px, 1fr)); }
.app-card, .submission-card { background-color: var(--input-bg); border: 1px solid var(--border-color); border-radius: 5px; padding: 15px; text-align: center; }
.submission-card { text-align: left; }
.submission-card h4 { margin: 0 0 5px 0; }
.submission-details { margin-top: 15px; display: flex; gap: 10px; }
.submission-actions { margin-top: 15px; display: flex; gap: 10px; }
.submission-actions button, .submission-details button { padding: 5px 10px; cursor: pointer; border: 1px solid var(--main-border); border-radius: 3px; background: var(--button-bg); color: var(--button-text); }
.approve-btn { background-color: #28a745 !important; color: white !important; border-color: #28a745 !important; }
.reject-btn { background-color: #dc3545 !important; color: white !important; border-color: #dc3545 !important; }
</style>
<div class="appstore-container">
    <nav class="appstore-nav">
        <div class="appstore-tab active" data-pane="browse-pane">Browse</div>
        </nav>
        <div class="appstore-content">
            <div id="browse-pane" class="appstore-pane active">
                <div class="app-grid">Loading applications...</div>
            </div>
            <div id="submissions-pane" class="appstore-pane">
                <div class="submission-list">Loading submissions...</div>
            </div>
        </div>
    </div>
    `;

    const nav = container.querySelector('.appstore-nav');
    const panes = container.querySelectorAll('.appstore-pane');
    let submissionsCache = [];

    // --- 2. Tab Switching Logic ---
    const _switchTab = (tabEl) => {
        container.querySelectorAll('.appstore-tab').forEach(t =>
            t.classList.remove('active'));
        panes.forEach(p => p.classList.remove('active'));
        tabEl.classList.add('active');

        container.querySelector(`#${tabEl.dataset.pane}`).classList.add('active');
    }
}
```

```
};

// --- 3. Core Rendering Functions ---
const _renderApps = () => {
    const grid = container.querySelector('#browse-pane .app-grid');
    grid.innerHTML = 'Loading applications...';
    this.kernel.api.apps.call('listApps')
        .then(apps => {
            grid.innerHTML = '';
            if (apps.length === 0) {
                grid.innerHTML = '<p>No applications installed.</p>';
                return;
            }
            apps.forEach(app => {
                const card = document.createElement('div');
                card.className = 'app-card';
                card.innerHTML = `<div class="icon">${app.icon || '📦'}</div><h4>${app.title}</h4><p>${app.description}</p>`;
                card.onclick = () =>
                    this.kernel.launchApplication(app.id);
                grid.appendChild(card);
            });
        })
        .catch(err => {
            grid.innerHTML = `<p style="color:red;">Error loading
apps: ${err.message}</p>`;
        });
};

const _renderSubmissions = () => {
    const list = container.querySelector('#submissions-pane
.submission-list');
    list.innerHTML = 'Loading submissions...';
    this.kernel.api.apps.call('listSubmissions')
        .then(submissions => {
            submissionsCache = submissions;
            list.innerHTML = '';
            if (submissions.length === 0) {
                list.innerHTML = '<p>No pending submissions.</p>';
                return;
            }
            submissions.forEach(sub => {
                const card = document.createElement('div');
                card.className = 'submission-card';
                card.dataset.id = sub.manifest.id;
                card.innerHTML =
                    `<h4>${sub.manifest.title} <small>
(v${sub.manifest.version})</small></h4>
<p><strong>ID:</strong> ${sub.manifest.id}</p>
<p>${sub.manifest.description} || 'No description
provided.'</p>
<p><small>Submitted by: ${sub.submitted_by}</small> on
${new Date(sub.submitted_at).toLocaleString()}</small></p>
<div class="submission-details">
`;
```

```
                <button class="button" data-action="view-
manifest">View Manifest</button>
                <button class="button" data-action="view-
code">View Code</button>
            </div>
            <div class="submission-actions">
                <button class="approve-btn" data-
action="approve">Approve</button>
                <button class="reject-btn" data-
action="reject">Reject</button>
            </div>
        `;
        list.appendChild(card);
    });
}
.catch(err => {
    list.innerHTML = `<p style="color:red;">Error loading
submissions: ${err.message}</p>`;
});
};

const showCodeDialog = (title, content) => {
    const pre = document.createElement('pre');
    pre.style.whiteSpace = 'pre-wrap';
    pre.style.wordBreak = 'break-all';
    pre.style.maxHeight = '400px';
    pre.style.overflowY = 'auto';
    pre.style.background = 'var(--terminal-bg)';
    pre.style.padding = '10px';
    pre.style.border = '1px solid var(--main-border)';
    pre.style.borderRadius = '3px';
    pre.textContent = content;

    this.kernel.modules.ui.showDialog(title, pre.outerHTML, { buttons:
['Copy', 'Close'] })
    .then(result => {
        if (result.button === 'Copy') {
            navigator.clipboard.writeText(content).then(() => {
                this.kernel.modules.ui.showNotification('Success',
'Content copied to clipboard.');
            }).catch(err => {
                this.kernel.modules.ui.showNotification('Error',
'Failed to copy content.', 5000);
            });
        }
    });
};

// --- 4. Admin-Specific UI and Event Handling ---
if (this.kernel.modules.security &&
this.kernel.modules.security.isAdmin()) {
    const submissionsTab = document.createElement('div');
    submissionsTab.className = 'appstore-tab';
    submissionsTab.dataset.pane = 'submissions-pane';
}
```

```
        submissionsTab.textContent = 'Submissions';
        nav.appendChild(submissionsTab);
    }

    container.querySelectorAll('.appstore-tab').forEach(tab => {
        tab.addEventListener('click', () => {
            _switchTab(tab);
            if (tab.dataset.pane === 'submissions-pane') {
                _renderSubmissions();
            }
        });
    });

    container.querySelector('#submissions-pane').addEventListener('click',
e => {
    const target = e.target;
    const action = target.dataset.action;
    const card = target.closest('.submission-card');
    if (!action || !card) return;

    const appId = card.dataset.id;
    const submission = submissionsCache.find(s => s.manifest.id ===
appId);
    if (!submission) {
        this.kernel.modules.ui.showNotification('Error', 'Could not
find submission data. Please refresh.', 5000);
        return;
    }

    switch (action) {
        case 'approve':
            this.kernel.api.apps.call('approveSubmission', { id: appId
})
                .then(() => {
                    this.kernel.modules.ui.showNotification('Success',
`App '${appId}' approved.`);
                    _renderSubmissions();
                    _renderApps();
                })
                .catch(err =>
this.kernel.modules.ui.showNotification('Error', err.message, 5000));
                break;
        case 'reject':
            this.kernel.api.apps.call('rejectSubmission', { id: appId
})
                .then(() => {
                    this.kernel.modules.ui.showNotification('Success',
`App '${appId}' rejected.`);
                    _renderSubmissions();
                })
                .catch(err =>
this.kernel.modules.ui.showNotification('Error', err.message, 5000));
                break;
        case 'view-manifest':
    }
})
```

```
        showCodeDialog('Manifest: ' + submission.manifest.title,
JSON.stringify(submission.manifest, null, 2));
        break;
    case 'view-code':
        showCodeDialog('Code: ' + submission.manifest.title,
submission.code);
        break;
    }
});

// --- 5. Initial Load ---
_renderApps();
}

/***
 * Load featured apps for the App Store
 * @private
 * @param {string} windowId - Window ID
 */
_loadFeaturedApps(windowId) {
    const contentDiv = document.getElementById(`#${windowId}-content-featured`);

    // Fetch featured apps from local storage or server
    // For now, we'll use some demo data
    const featuredApps = [
        {
            id: 'calculator',
            title: 'Scientific Calculator',
            description: 'Advanced calculator with scientific functions',
            icon: '💻',
            author: 'GOS Team',
            version: '1.0.0',
            category: 'Utilities',
            featured: true,
            banner: '📝 Featured App'
        },
        {
            id: 'notes',
            title: 'Quick Notes',
            description: 'Simple note-taking application with markdown support',
            icon: '📝',
            author: 'GOS Team',
            version: '1.0.0',
            category: 'Productivity',
            featured: true,
            banner: '⭐ Staff Pick'
        }
    ];

    // Create HTML content
}
```

```
let html = `

    <div class="featured-banner" style="background: linear-
gradient(135deg, var(--highlight), var(--main-border)); color: var(--main-bg);
padding: 20px; border-radius: 8px; margin-bottom: 20px; text-align: center;">
        <h2 style="margin-top: 0;">Welcome to the Text Reversion App
Store</h2>
        <p>Discover apps, share your creations, and join the
community</p>
    </div>

    <h3>Featured Apps</h3>
    <div class="app-grid" style="display: grid; grid-template-columns:
repeat(auto-fill, minmax(280px, 1fr)); gap: 20px;">
`;

// Add app cards for featured apps
featuredApps.forEach(app => {
    html += this._createAppCard(app);
});

html += `
    </div>

    <div style="margin-top: 30px; text-align: center;">
        <h3>Develop Your Own Apps</h3>
        <p>Create apps using the built-in development tools and share
them with the community</p>
        <button id="${windowId}-goto-develop" class="button button-
primary" style="margin-top: 10px;">Start Developing</button>
    </div>
`;

// Set content
contentDiv.innerHTML = html;

// Add event listeners - AFTER the HTML has been added to DOM
// Use requestAnimationFrame to ensure DOM is updated
requestAnimationFrame(() => {
    // First check if the develop button exists
    const developButton = document.getElementById(`#${windowId}-goto-
develop`);

    if (developButton) {
        developButton.addEventListener('click', () => {
            // Switch to develop tab
            const developTab = document.getElementById(`#${windowId}-
tab-develop`);

            if (developTab) {
                developTab.click();
            }
        });
    }

    // Add event listeners for app buttons
    featuredApps.forEach(app => {
```

```
const installButton = document.getElementById(`#${windowId}-install-${app.id}`);
if (installButton) {
    installButton.addEventListener('click', (e) => {
        e.stopPropagation();
        this._installApp(app);
    });
}

const appCard = document.getElementById(`#${windowId}-app-${app.id}`);
if (appCard) {
    appCard.addEventListener('click', () => {
        this._showAppDetails(app, windowId);
    });
}
});

/**
 * Load all apps for the App Store
 * @private
 * @param {string} windowId - Window ID
 */
_loadAllApps(windowId) {
    const contentDiv = document.getElementById(`#${windowId}-content-all`);
    if (!contentDiv) {
        console.error(`Content div not found: ${windowId}-content-all`);
        return;
    }

    // Fetch all apps from local storage or server
    // For now, we'll use some demo data
    const allApps = [
        {
            id: 'calculator',
            title: 'Scientific Calculator',
            description: 'Advanced calculator with scientific functions',
            icon: '💻',
            author: 'GOS Team',
            version: '1.0.0',
            category: 'Utilities'
        },
        {
            id: 'notes',
            title: 'Quick Notes',
            description: 'Simple note-taking application with markdown support',
            icon: '📝',
            author: 'GOS Team',
            version: '1.0.0',
            category: 'Productivity'
        },
    ];
}
```

```
{
    id: 'calendar',
    title: 'Calendar',
    description: 'Schedule and manage your events',
    icon: '📅',
    author: 'GOS Team',
    version: '1.0.0',
    category: 'Productivity'
},
{
    id: 'weather',
    title: 'Weather',
    description: 'Check the weather in your area',
    icon: '🌤',
    author: 'GOS Team',
    version: '1.0.0',
    category: 'Utilities'
},
{
    id: 'games',
    title: 'Game Collection',
    description: 'A collection of simple games',
    icon: '🎮',
    author: 'Community',
    version: '1.0.0',
    category: 'Games'
}
];

// Create HTML content
let html = `
<h3>Browse All Apps</h3>

<div class="category-filter" style="margin-bottom: 20px;">
    <label>Category: </label>
    <select id="${windowId}-category-filter" class="settings-select" style="width: auto; display: inline-block;">
        <option value="All">All Categories</option>
        <option value="Productivity">Productivity</option>
        <option value="Development">Development</option>
        <option value="Science">Science</option>
        <option value="System">System</option>
        <option value="Games">Games</option>
        <option value="Utilities">Utilities</option>
    </select>
</div>

<div id="${windowId}-app-grid" class="app-grid" style="display: grid; grid-template-columns: repeat(auto-fill, minmax(280px, 1fr)); gap: 20px;">
`;

// Add app cards for all apps
allApps.forEach(app => {
    html += this._createAppCard(app);
}
```

```
});

html += `</div>`;

// Set content first
contentDiv.innerHTML = html;

// Use requestAnimationFrame to ensure DOM is updated before adding
event listeners
requestAnimationFrame(() => {
    // Add event listener for category filter
    const categoryFilter = document.getElementById(`#${windowId}-
category-filter`);
    if (categoryFilter) {
        categoryFilter.addEventListener('change', (e) => {
            const category = e.target.value;

            // Filter apps by category
            const filteredApps = category === 'All'
                ? allApps
                : allApps.filter(app => app.category === category);

            // Update app grid
            const appGrid = document.getElementById(`#${windowId}-app-
grid`);
            if (!appGrid) return;

            appGrid.innerHTML = '';

            // Add filtered apps to the grid
            filteredApps.forEach(app => {
                appGrid.innerHTML += this._createAppCard(app);
            });

            // Re-add event listeners after updating the grid content
            // Need to use another requestAnimationFrame to ensure DOM
is updated
            requestAnimationFrame(() => {
                // Add event listeners for newly created buttons
                filteredApps.forEach(app => {
                    const installButton =
document.getElementById(`#${windowId}-install-${app.id}`);
                    if (installButton) {
                        installButton.addEventListener('click', (e) =>
{
                            e.stopPropagation();
                            this._installApp(app);
                        });
                    }
                });
            });
        });
    }
});
```

```
        this._showAppDetails(app, windowId);
    });
}
});
});
});
}

// Add event listeners for install buttons and app cards
allApps.forEach(app => {
    const installButton = document.getElementById(`#${windowId}-install-${app.id}`);
    if (installButton) {
        installButton.addEventListener('click', (e) => {
            e.stopPropagation();
            this._installApp(app);
        });
    }

    const appCard = document.getElementById(`#${windowId}-app-${app.id}`);
    if (appCard) {
        appCard.addEventListener('click', () => {
            this._showAppDetails(app, windowId);
        });
    }
});

/**
 * Load installed apps for the App Store
 * @private
 * @param {string} windowId - Window ID
 */
_loadInstalledApps(windowId) {
    const contentDiv = document.getElementById(`#${windowId}-content-installed`);
    if (!contentDiv) {
        console.error(`Content div not found: ${windowId}-content-installed`);
        return;
    }

    // Get installed apps from local storage
    const installedApps =
JSON.parse(localStorage.getItem('gos_installed_apps') || '[]');

    // Create HTML content
    let html = `
        <h3>My Installed Apps</h3>
    `;

    if (installedApps.length === 0) {
```

```
        html += `

            <div style="text-align: center; padding: 40px; opacity: 0.7;">
                <p>You haven't installed any apps yet</p>
                <button id="${windowId}-browse-apps" class="button button-primary" style="margin-top: 10px;">Browse Apps</button>
            </div>
        `;
    } else {
        html += `

            <div class="app-grid" style="display: grid; grid-template-columns: repeat(auto-fill, minmax(280px, 1fr)); gap: 20px;">
        `;

        // Add app cards for installed apps
        installedApps.forEach(app => {
            html += this._createAppCard(app, true);
        });

        html += `</div>`;
    }

    // Set content first
    contentDiv.innerHTML = html;

    // Use requestAnimationFrame to ensure DOM is updated before adding event listeners
    requestAnimationFrame(() => {
        if (installedApps.length === 0) {
            // Handle the empty state with "Browse Apps" button
            const browseAppsButton = document.getElementById(`${windowId}-browse-apps`);

            if (browseAppsButton) {
                browseAppsButton.addEventListener('click', () => {
                    // Switch to all apps tab
                    const allAppsTab =
document.getElementById(`${windowId}-tab-all`);
                    if (allAppsTab) {
                        allAppsTab.click();
                    } else {
                        console.warn(`All apps tab element not found: ${windowId}-tab-all`);
                    }
                });
            }
        } else {
            // Add event listeners for app cards and uninstall buttons
            installedApps.forEach(app => {
                // Add event listener for uninstall button
                const uninstallButton =
document.getElementById(`${windowId}-uninstall-${app.id}`);
                if (uninstallButton) {
                    uninstallButton.addEventListener('click', (e) => {
                        e.stopPropagation(); // Prevent triggering the app card click
                    });
                }
            });
        }
    });
}
```

```
        this._uninstallApp(app.id, windowId);
    });
}

// Add event listener for app card click
const appCard = document.getElementById(`#${windowId}-app-${app.id}`);
if (appCard) {
    appCard.addEventListener('click', () => {
        this._showAppDetails(app, windowId);
    });
}
});

/**
 * Create an app card HTML
 * @private
 * @param {Object} app - App data
 * @param {boolean} [installed=false] - Whether the app is installed
 * @returns {string} App card HTML
 */
_createAppCard(app, installed = false) {
    // Create banner HTML if app has a banner
    const bannerHtml = app.banner
        ? `<div style="background: var(--highlight); color: var(--main-bg); padding: 5px 15px; font-size: 12px;">${app.banner}</div>` :
        '';

    // Create button HTML based on installed status
    const buttonHtml = installed
        ? `<button id="${this.kernel.modules.process.windowId}-uninstall-${app.id}" class="button">Uninstall</button>` :
        `<button id="${this.kernel.modules.process.windowId}-install-${app.id}" class="button button-primary">Install</button>`;

    return `
        <div id="${this.kernel.modules.process.windowId}-app-${app.id}" class="app-card" style="background-color: var(--terminal-bg); border-radius: 8px; overflow: hidden; cursor: pointer; transition: transform 0.2s; height: 100%;">
            ${bannerHtml}
            <div style="padding: 20px;">
                <div style="display: flex; align-items: center; margin-bottom: 15px;">
                    <div style="font-size: 32px; margin-right: 15px;">${app.icon}</div>
                    <div>
                        <h3 style="margin: 0 0 5px 0;">${app.title}</h3>
                        <div style="font-size: 12px; opacity: 0.7;">by ${app.author} | v${app.version}</div>
                    </div>
                </div>
            </div>
        </div>
    `;
}
```

```
        <p style="margin-bottom: 15px; height: 40px; overflow: hidden; display: -webkit-box; -webkit-line-clamp: 2; -webkit-box-orient: vertical;">${app.description}</p>

            <div style="display: flex; justify-content: space-between; align-items: center;">
                <div style="font-size: 12px; background-color: rgba(var(--highlight), 0.1); padding: 3px 8px; border-radius: 12px;">${app.category}</div>
                ${buttonHtml}
            </div>
        </div>
    </div>
`;
}

/***
 * Initialize the develop tab for the App Store
 * @private
 * @param {string} windowId - Window ID
 */
_initializeDevelopTab(windowId) {
    const contentDiv = document.getElementById(`#${windowId}-content-develop`);
    if (!contentDiv) {
        console.error(`Content div not found: ${windowId}-content-develop`);
        return;
    }

    // Default code template with proper escaping
    const defaultCode = `{
// App initialization code
initialize: function(container, process, appInfo) {
    // Create app UI
    container.innerHTML = `
        <div style="padding: 20px; text-align: center;">
            <h2>${appInfo.title}</h2>
            <p>Hello, World!</p>
        </div>
`;
    // Add your app logic here
}
`;

    // Create HTML content
    contentDiv.innerHTML = `
        <h3>Develop Your Own App</h3>

        <div class="app-development-container" style="display: flex; flex-direction: column; gap: 20px;">
            <div class="app-metadata" style="background-color: var(--
```

```
terminal-bg); padding: 20px; border-radius: 8px;">
    <h4 style="margin-top: 0;">App Metadata</h4>

    <div style="display: grid; grid-template-columns: 1fr 1fr;
gap: 15px;">
        <div class="settings-option">
            <label class="settings-label">App ID (unique, no
spaces)</label>
            <input type="text" id="${windowId}-app-id"
class="settings-input" placeholder="my-awesome-app">
        </div>

        <div class="settings-option">
            <label class="settings-label">App Title</label>
            <input type="text" id="${windowId}-app-title"
class="settings-input" placeholder="My Awesome App">
        </div>

        <div class="settings-option">
            <label class="settings-label">Icon (emoji)</label>
            <input type="text" id="${windowId}-app-icon"
class="settings-input" placeholder="🔗">
        </div>

        <div class="settings-option">
            <label class="settings-label">Category</label>
            <select id="${windowId}-app-category"
class="settings-select">
                <option
value="Productivity">Productivity</option>
                <option
value="Development">Development</option>
                <option value="Science">Science</option>
                <option value="System">System</option>
                <option value="Games">Games</option>
                <option value="Utilities">Utilities</option>
            </select>
        </div>

        <div class="settings-option">
            <label class="settings-label">Author</label>
            <input type="text" id="${windowId}-app-author"
class="settings-input" placeholder="Your Name">
        </div>

        <div class="settings-option">
            <label class="settings-label">Version</label>
            <input type="text" id="${windowId}-app-version"
class="settings-input" placeholder="1.0.0" value="1.0.0">
        </div>
    </div>

    <div class="settings-option" style="margin-top: 15px;">
        <label class="settings-label">Description</label>
```

```
        <textarea id="${windowId}-app-description"
class="settings-input" style="height: 80px; resize: vertical;
placeholder="Describe your app here..."></textarea>
    </div>

    <div class="settings-option" style="margin-top: 15px;">
        <label class="settings-label">Required
    Permissions</label>
        <div style="display: flex; flex-wrap: wrap; gap: 10px;
margin-top: 5px;">
            <label style="display: flex; align-items:
center;">
                <input type="checkbox" id="${windowId}-perm-
filesystem" checked>
                <span style="margin-left: 5px;">Filesystem
Access</span>
            </label>
            <label style="display: flex; align-items:
center;">
                <input type="checkbox" id="${windowId}-perm-
network">
                <span style="margin-left: 5px;">Network
Access</span>
            </label>
            <label style="display: flex; align-items:
center;">
                <input type="checkbox" id="${windowId}-perm-
system">
                <span style="margin-left: 5px;">System
Access</span>
            </label>
        </div>
    </div>

    <div class="app-code" style="background-color: var(--terminal-
bg); padding: 20px; border-radius: 8px;">
        <h4 style="margin-top: 0;">App Code</h4>
        <p style="opacity: 0.7; margin-bottom: 15px;">Write your
app code in JavaScript. This code will be executed when your app is launched.</p>

        <textarea id="${windowId}-app-code" style="width: 100%;
height: 300px; background-color: var(--terminal-bg); color: var(--main-text);
border: 1px solid var(--main-border); font-family: 'Courier New', monospace; font-
size: 14px; padding: 10px; resize: vertical; tab-size: 2;"></textarea>
    </div>

    <div class="app-actions" style="display: flex; gap: 10px;
justify-content: space-between;">
        <div>
            <button id="${windowId}-clear-app"
class="button">Clear Form</button>
            <button id="${windowId}-test-app" class="button">Test
App</button>
        </div>
    </div>

```

```
        </div>
        <div>
            <button id="${windowId}-export-app"
class="button">Export as JSON</button>
            <button id="${windowId}-save-app" class="button
button-primary">Save App</button>
        </div>
    </div>

        <div class="app-json-preview" style="background-color: var(--terminal-bg); padding: 20px; border-radius: 8px; display: none;">
            <h4 style="margin-top: 0;">App JSON Preview</h4>
            <pre id="${windowId}-app-json" style="white-space: pre-wrap; word-break: break-all; overflow-x: auto; background-color: var(--terminal-bg); padding: 10px; border: 1px solid var(--main-border); border-radius: 4px; font-family: 'Courier New', monospace; font-size: 12px;"></pre>
        <div style="margin-top: 15px;">
            <button id="${windowId}-copy-json" class="button">Copy
JSON</button>
            <button id="${windowId}-close-preview"
class="button">Close Preview</button>
        </div>
    </div>

        <div class="app-import" style="background-color: var(--terminal-bg); padding: 20px; border-radius: 8px;">
            <h4 style="margin-top: 0;">Import App from JSON</h4>
            <p style="opacity: 0.7; margin-bottom: 15px;">Paste the
app JSON here to import an existing app.</p>
            <textarea id="${windowId}-import-json" style="width: 100%; height: 100px; background-color: var(--terminal-bg); color: var(--main-text); border: 1px solid var(--main-border); font-family: 'Courier New', monospace; font-size: 14px; padding: 10px; resize: vertical;" placeholder='{"id": "app-id",
"title": "App Title", ...}'></textarea>
        <div style="margin-top: 10px;">
            <button id="${windowId}-import-app"
class="button">Import App</button>
        </div>
    </div>
`;

// Use requestAnimationFrame to ensure DOM is updated before
interacting with elements
requestAnimationFrame(() => {
    // Set default code after HTML is in DOM
    const codeTextarea = document.getElementById(`#${windowId}-app-
code`);
    if (codeTextarea) {
        codeTextarea.value = defaultCode;
    }
}
```

```
// Add event listeners for each button
const clearAppBtn = document.getElementById(`#${windowId}-clear-app`);
if (clearAppBtn) {
    clearAppBtn.addEventListener('click', () => {
        // Get form elements with null checks
        const appIdInput = document.getElementById(`#${windowId}-app-id`);
        const appTitleInput =
document.getElementById(`#${windowId}-app-title`);
        const appIconInput = document.getElementById(`#${windowId}-app-icon`);
        const appDescInput = document.getElementById(`#${windowId}-app-description`);
        const appAuthorInput =
document.getElementById(`#${windowId}-app-author`);
        const appVersionInput =
document.getElementById(`#${windowId}-app-version`);
        const appCodeTextarea =
document.getElementById(`#${windowId}-app-code`);

        // Clear form if elements exist
        if (appIdInput) appIdInput.value = '';
        if (appTitleInput) appTitleInput.value = '';
        if (appIconInput) appIconInput.value = '';
        if (appDescInput) appDescInput.value = '';
        if (appAuthorInput) appAuthorInput.value = '';
        if (appVersionInput) appVersionInput.value = '1.0.0';
        if (appCodeTextarea) appCodeTextarea.value = defaultCode;
    });
}

const testAppBtn = document.getElementById(`#${windowId}-test-app`);
if (testAppBtn) {
    testAppBtn.addEventListener('click', () => {
        // Validate form
        const appIdInput = document.getElementById(`#${windowId}-app-id`);
        const appTitleInput =
document.getElementById(`#${windowId}-app-title`);

        if (!appIdInput || !appTitleInput) {
            console.error('Required form elements not found');
            return;
        }

        const appId = appIdInput.value;
        const appTitle = appTitleInput.value;

        if (!appId || !appTitle) {
            this.kernel.modules.ui.showNotification('Error', 'App ID and Title are required', 3000);
        }
    });
}
```

```
        return;
    }

    // Get app data
    const appData = this._getAppDataFromForm(windowId);
    if (!appData) {
        this.kernel.modules.ui.showNotification('Error',
'Failed to get app data', 3000);
        return;
    }

    // Show notification
    this.kernel.modules.ui.showNotification('Testing',
`Testing app: ${appTitle}`, 2000);
};

}

const exportAppBtn = document.getElementById(`#${windowId}-export-
app`);
if (exportAppBtn) {
    exportAppBtn.addEventListener('click', () => {
        // Validate form
        const appIdInput = document.getElementById(`#${windowId}-
app-id`);
        const appTitleInput =
document.getElementById(`#${windowId}-app-title`);

        if (!appIdInput || !appTitleInput) {
            console.error('Required form elements not found');
            return;
        }

        const appId = appIdInput.value;
        const appTitle = appTitleInput.value;

        if (!appId || !appTitle) {
            this.kernel.modules.ui.showNotification('Error', 'App
ID and Title are required', 3000);
            return;
        }

        // Get app data
        const appData = this._getAppDataFromForm(windowId);
        if (!appData) {
            this.kernel.modules.ui.showNotification('Error',
'Failed to get app data', 3000);
            return;
        }

        // Show JSON preview
        const jsonPreElement =
document.getElementById(`#${windowId}-app-json`);
        const jsonPreviewContainer =
document.querySelector(`#${windowId}-content-develop .app-json-preview`);
```

```
        if (jsonPreElement && jsonPreviewContainer) {
            jsonPreElement.textContent = JSON.stringify(appData,
null, 2);
            jsonPreviewContainer.style.display = 'block';
        }
    });

const copyJsonBtn = document.getElementById(`#${windowId}-copy-
json`);
if (copyJsonBtn) {
    copyJsonBtn.addEventListener('click', () => {
        const jsonPreElement =
document.getElementById(`#${windowId}-app-json`);
        if (!jsonPreElement) return;

        const json = jsonPreElement.textContent;

        // Copy to clipboard
        navigator.clipboard.writeText(json).then(() => {
            this.kernel.modules.ui.showNotification('Success',
'JSON copied to clipboard', 2000);
        }).catch(err => {
            this.kernel.modules.ui.showNotification('Error',
'Failed to copy JSON', 3000);
        });
    });
}

const closePreviewBtn = document.getElementById(`#${windowId}-
close-preview`);
if (closePreviewBtn) {
    closePreviewBtn.addEventListener('click', () => {
        const jsonPreviewContainer =
document.querySelector(`#${windowId}-content-develop .app-json-preview`);
        if (jsonPreviewContainer) {
            jsonPreviewContainer.style.display = 'none';
        }
    });
}

const saveAppBtn = document.getElementById(`#${windowId}-save-
app`);
if (saveAppBtn) {
    saveAppBtn.addEventListener('click', () => {
        // Validate form
        const appIdInput = document.getElementById(`#${windowId}-
app-id`);
        const appTitleInput =
document.getElementById(`#${windowId}-app-title`);

        if (!appIdInput || !appTitleInput) {
            console.error('Required form elements not found');
        }
    });
}
```

```
        return;
    }

    const appId = appIdInput.value;
    const appTitle = appTitleInput.value;

    if (!appId || !appTitle) {
        this.kernel.modules.ui.showNotification('Error', 'App
ID and Title are required', 3000);
        return;
    }

    // Get app data
    const appData = this._getAppDataFromForm(windowId);
    if (!appData) {
        this.kernel.modules.ui.showNotification('Error',
'Failed to get app data', 3000);
        return;
    }

    // Save app
    this._saveApp(appData);

    // Show notification
    this.kernel.modules.ui.showNotification('Success', `App
"${appTitle}" saved successfully`, 3000);
};

}

const importAppBtn = document.getElementById(`#${windowId}-import-
app`);
if (importAppBtn) {
    importAppBtn.addEventListener('click', () => {
        const importJsonTextarea =
document.getElementById(`#${windowId}-import-json`);
        if (!importJsonTextarea) return;

        const jsonText = importJsonTextarea.value;

        if (!jsonText) {
            this.kernel.modules.ui.showNotification('Error', 'No
JSON data provided', 3000);
            return;
        }

        try {
            const appData = JSON.parse(jsonText);

            // Validate app data
            if (!appData.id || !appData.title) {
                this.kernel.modules.ui.showNotification('Error',
'Invalid app data', 3000);
                return;
            }
        }
    });
}
```

```
// Get all form elements
const appIdInput =
document.getElementById(`#${windowId}-app-id`);
const appTitleInput =
document.getElementById(`#${windowId}-app-title`);
const appIconInput =
document.getElementById(`#${windowId}-app-icon`);
const appDescInput =
document.getElementById(`#${windowId}-app-description`);
const appAuthorInput =
document.getElementById(`#${windowId}-app-author`);
const appVersionInput =
document.getElementById(`#${windowId}-app-version`);
const appCategorySelect =
document.getElementById(`#${windowId}-app-category`);
const permFileCheck =
document.getElementById(`#${windowId}-perm-filesystem`);
const permNetworkCheck =
document.getElementById(`#${windowId}-perm-network`);
const permSystemCheck =
document.getElementById(`#${windowId}-perm-system`);
const appCodeTextarea =
document.getElementById(`#${windowId}-app-code`);

// Fill form with app data if elements exist
if (appIdInput) appIdInput.value = appData.id;
if (appTitleInput) appTitleInput.value =
appData.title;
if (appIconInput) appIconInput.value = appData.icon ||
 '';
if (appDescInput) appDescInput.value =
appData.description || '';
if (appAuthorInput) appAuthorInput.value =
appData.author || '';
if (appVersionInput) appVersionInput.value =
appData.version || '1.0.0';
if (appCategorySelect) appCategorySelect.value =
appData.category || 'Utilities';

// Set permissions
if (appData.permissions) {
    if (permFileCheck) permFileCheck.checked =
appData.permissions.includes('filesystem');
    if (permNetworkCheck) permNetworkCheck.checked =
appData.permissions.includes('network');
    if (permSystemCheck) permSystemCheck.checked =
appData.permissions.includes('system');
}

// Set code
if (appCodeTextarea && appData.code) {
    appCodeTextarea.value = typeof appData.code ===
'string'
```

```
        ? appData.code
        : JSON.stringify(appData.code, null, 2);
    }

    // Show notification
    this.kernel.modules.ui.showNotification('Success',
`App "${appData.title}" imported successfully`, 3000);
} catch (error) {
    this.kernel.modules.ui.showNotification('Error',
`Failed to parse JSON: ${error.message}`, 5000);
}
});

}

/**
 * Initialize the community tab for the App Store
 * @private
 * @param {string} windowId - Window ID
 */
_initializeCommunityTab(windowId) {
    const contentDiv = document.getElementById(`#${windowId}-content-
community`);
    if (!contentDiv) {
        console.error(`Content div not found: ${windowId}-content-
community`);
        return;
    }

    // Create HTML content
    contentDiv.innerHTML =
        `

<div class="community-section">
                <h3>Community Support</h3>

                <div style="background-color: var(--terminal-bg); padding: 20px; border-radius: 8px;">
                    <p>Get support and share your apps with the Text Reversion community!</p>
                <h4 style="margin-top: 20px;">Submit Your App for Review</h4>
                    <p>Have you created an app you'd like to share with the community? Submit it for review and it could be featured in the App Store!</p>
                <div style="background-color: rgba(var(--highlight), 0.1); padding: 15px; border-radius: 4px; margin-top: 15px;">
                    <p><strong>Email your app JSON file to:</strong></p>
                <div>
                    <p style="font-size: 18px; margin-top: 5px;">appsubmissions@genesis-os.example.com</p>
                </div>
            </div>

`;
```



```
20px; border-radius: 8px;">>  
    <p>Join the Text Reversion community and help shape  
the future of the platform!</p>  
  
    <div style="display: grid; grid-template-columns:  
repeat(auto-fit, minmax(200px, 1fr)); gap: 20px; margin-top: 20px;">  
        <div id="${windowId}-forum-card" style="text-  
align: center; padding: 15px; border: 1px solid var(--main-border); border-radius:  
8px; cursor: pointer;">  
            <div style="font-size: 32px; margin-bottom:  
10px;">🔗</div>  
            <h4 style="margin: 0 0 10px 0;">Join the  
Forum</h4>  
            <p>Discuss ideas and get help from other  
users</p>  
        </div>  
  
        <div id="${windowId}-contribute-card" style="text-  
align: center; padding: 15px; border: 1px solid var(--main-border); border-radius:  
8px; cursor: pointer;">  
            <div style="font-size: 32px; margin-bottom:  
10px;">💡</div>  
            <h4 style="margin: 0 0 10px  
0;">Contribute</h4>  
            <p>Help improve the platform with your  
skills</p>  
        </div>  
  
        <div id="${windowId}-docs-card" style="text-align:  
center; padding: 15px; border: 1px solid var(--main-border); border-radius: 8px;  
cursor: pointer;">  
            <div style="font-size: 32px; margin-bottom:  
10px;">📄</div>  
            <h4 style="margin: 0 0 10px  
0;">Documentation</h4>  
            <p>Learn how to make the most of Text  
Reversion</p>  
        </div>  
    </div>  
  </div>  
  
<div class="community-section">  
  <h3>Contact Us</h3>  
  
  <div style="background-color: var(--terminal-bg); padding:  
20px; border-radius: 8px;">  
    <p>Have questions or suggestions? We'd love to hear  
from you!</p>  
  
    <form id="${windowId}-contact-form" style="margin-top:  
20px;">  
      <div style="margin-bottom: 15px;">  
        <label style="display: block; margin-bottom:  
10px;">
```

```
5px;">>Your Name</label>
      <input type="text" id="${windowId}-contact-
name" style="width: 100%; padding: 8px; background-color: var(--terminal-bg);
color: var(--main-text); border: 1px solid var(--main-border); border-radius:
4px;">
      </div>

      <div style="margin-bottom: 15px;">
        <label style="display: block; margin-bottom:
5px;">Email Address</label>
          <input type="email" id="${windowId}-contact-
email" style="width: 100%; padding: 8px; background-color: var(--terminal-bg);
color: var(--main-text); border: 1px solid var(--main-border); border-radius:
4px;">
          </div>

          <div style="margin-bottom: 15px;">
            <label style="display: block; margin-bottom:
5px;">Message</label>
              <textarea id="${windowId}-contact-message"
style="width: 100%; height: 120px; padding: 8px; background-color: var(--terminal-
bg); color: var(--main-text); border: 1px solid var(--main-border); border-radius:
4px; resize: vertical;"></textarea>
            </div>

            <button id="${windowId}-contact-submit"
type="button" class="button button-primary" style="width: 100%;">Send
Message</button>
          </form>
        </div>
      </div>
    </div>
  `;
```

// Use requestAnimationFrame to ensure DOM is updated before adding event listeners

```
requestAnimationFrame(() => {
  // Add event listeners for cards in the "Get Involved" section
  const forumCard = document.getElementById(`#${windowId}-forum-
card`);

  if (forumCard) {
    forumCard.addEventListener('click', () => {
      this.kernel.modules.ui.showNotification('Forum', 'Opening
forum in a new window...', 2000);
      // In a real implementation, this might open a URL or
      launch a forum app
    });
  }

  // Add hover effect
  forumCard.addEventListener('mouseenter', () => {
    forumCard.style.backgroundColor = 'var(--highlight)';
    forumCard.style.color = 'var(--main-bg)';
  });
});
```

```
        forumCard.addEventListener('mouseleave', () => {
            forumCard.style.backgroundColor = '';
            forumCard.style.color = '';
        });
    }

    const contributeCard = document.getElementById(`#${windowId}-contribute-card`);
    if (contributeCard) {
        contributeCard.addEventListener('click', () => {
            this.kernel.modules.ui.showNotification('Contribute',
'Opening contribution guidelines...', 2000);
            // In a real implementation, this might open guidelines or
a GitHub page
        });

        // Add hover effect
        contributeCard.addEventListener('mouseenter', () => {
            contributeCard.style.backgroundColor = 'var(--highlight)';
            contributeCard.style.color = 'var(--main-bg)';
        });

        contributeCard.addEventListener('mouseleave', () => {
            contributeCard.style.backgroundColor = '';
            contributeCard.style.color = '';
        });
    }

    const docsCard = document.getElementById(`#${windowId}-docs-card`);
    if (docsCard) {
        docsCard.addEventListener('click', () => {
            this.kernel.modules.ui.showNotification('Documentation',
'Opening documentation...', 2000);
            // In a real implementation, this might open the
documentation
        });

        // Add hover effect
        docsCard.addEventListener('mouseenter', () => {
            docsCard.style.backgroundColor = 'var(--highlight)';
            docsCard.style.color = 'var(--main-bg)';
        });

        docsCard.addEventListener('mouseleave', () => {
            docsCard.style.backgroundColor = '';
            docsCard.style.color = '';
        });
    }

    // Add event listener for contact form submission
    const contactSubmitBtn = document.getElementById(`#${windowId}-contact-submit`);
    if (contactSubmitBtn) {
        contactSubmitBtn.addEventListener('click', () => {
```

```
// Get form elements
const nameInput = document.getElementById(`#${windowId}-contact-name`);
const emailInput = document.getElementById(`#${windowId}-contact-email`);
const messageInput = document.getElementById(`#${windowId}-contact-message`);

if (!nameInput || !emailInput || !messageInput) {
    console.error('Contact form elements not found');
    return;
}

// Get form values
const name = nameInput.value.trim();
const email = emailInput.value.trim();
const message = messageInput.value.trim();

// Validate form
if (!name || !email || !message) {
    this.kernel.modules.ui.showNotification('Error',
'Please fill out all fields', 3000);
    return;
}

// Simple email validation
const emailRegex = /^[^@\s]+@[^\s@]+\.[^\s@]+$/;
if (!emailRegex.test(email)) {
    this.kernel.modules.ui.showNotification('Error',
'Please enter a valid email address', 3000);
    return;
}

// In a real implementation, this would send the message
to a server
// For now, just show a success notification
this.kernel.modules.ui.showNotification('Success', 'Your
message has been sent!', 3000);

// Clear form
nameInput.value = '';
emailInput.value = '';
messageInput.value = '';
});

}

/**
 * Show app details
 * @private
 * @param {Object} app - App data
 * @param {string} windowId - Window ID
 */
```

```
_showAppDetails(app, windowId) {
    // Create dialog
    const backdrop = document.createElement('div');
    backdrop.className = 'modal-backdrop';
    backdrop.style.zIndex = '2000';

    const dialog = document.createElement('div');
    dialog.className = 'modal';
    dialog.style.width = '600px';

    // Create buttons HTML based on app installation status
    const buttonsHtml = this._isAppInstalled(app.id)
        ? `<button class="button" data-
action="uninstall">Uninstall</button>
            <button class="button button-primary" data-
action="launch">Launch</button>
            : `<button class="button button-primary" data-
action="install">Install</button>`;

    // Build dialog content
    dialog.innerHTML = `
        <div class="modal-header">
            <div class="modal-title">${app.title}</div>
        </div>
        <div class="modal-body">
            <div style="display: flex; margin-bottom: 20px;">
                <div style="font-size: 48px; margin-right:
20px;">${app.icon}</div>
                <div>
                    <h3 style="margin: 0 0 5px 0;">${app.title}</h3>
                    <div style="font-size: 14px; opacity: 0.7;">by
${app.author} | v${app.version}</div>
                    <div style="font-size: 14px; margin-top:
5px;">Category: ${app.category}</div>
                </div>
            </div>

            <div style="margin-bottom: 20px;">
                <h4>Description</h4>
                <p>${app.description}</p>
            </div>

            <div>
                <h4>Details</h4>
                <table style="width: 100%; border-collapse: collapse;">
                    <tr>
                        <td style="padding: 8px 0; border-bottom: 1px
solid var(--main-border); width: 120px;">App ID</td>
                        <td style="padding: 8px 0; border-bottom: 1px
solid var(--main-border);">${app.id}</td>
                    </tr>
                    <tr>
                        <td style="padding: 8px 0; border-bottom: 1px
solid var(--main-border);">Version</td>
```

```
        <td style="padding: 8px 0; border-bottom: 1px solid var(--main-border);">${app.version}</td>
        </tr>
        <tr>
            <td style="padding: 8px 0; border-bottom: 1px solid var(--main-border);">Author</td>
            <td style="padding: 8px 0; border-bottom: 1px solid var(--main-border);">${app.author}</td>
        </tr>
        <tr>
            <td style="padding: 8px 0;">Permissions</td>
            <td style="padding: 8px 0;">${app.permissions ? app.permissions.join(', ') : 'Basic access'}</td>
        </tr>
    </table>
</div>
</div>
<div class="modal-footer">
    <button class="button" data-action="close">Close</button>
    ${buttonsHtml}
</div>
`;

// Add to DOM
backdrop.appendChild(dialog);
document.body.appendChild(backdrop);

// Show dialog with animation
setTimeout(() => {
    backdrop.classList.add('show');
}, 10);

// Set up button handlers
dialog.querySelector('button[data-
action="close"]').addEventListener('click', () => {
    backdrop.classList.remove('show');
    setTimeout(() => {
        backdrop.remove();
    }, 300);
});

if (this._isAppInstalled(app.id)) {
    dialog.querySelector('button[data-
action="uninstall"]').addEventListener('click', () => {
        this._uninstallApp(app.id, windowId);
        backdrop.classList.remove('show');
        setTimeout(() => {
            backdrop.remove();
        }, 300);
    });
}

dialog.querySelector('button[data-
action="launch"]').addEventListener('click', () => {
    this._launchApp(app.id);
});
```

```
        backdrop.classList.remove('show');
        setTimeout(() => {
            backdrop.remove();
        }, 300);
    });
} else {
    dialog.querySelector('button[data-
action="install"]').addEventListener('click', () => {
        this._installApp(app);
        backdrop.classList.remove('show');
        setTimeout(() => {
            backdrop.remove();
        }, 300);
    });
}
}

/**
 * Install an app
 * @private
 * @param {Object} app - App data
 */
_installApp(app) {
    // Check if app is already installed
    if (this._isAppInstalled(app.id)) {
        this.kernel.modules.ui.showNotification('Info', `${app.title} is
already installed`, 3000);
        return;
    }

    // Get installed apps from local storage
    const installedApps =
JSON.parse(localStorage.getItem('gos_installed_apps') || '[]');

    // Prepare app for installation
    const appToInstall = {
        ...app,
        // Add required properties for system compatibility
        window: app.window || {
            width: 800,
            height: 600,
            resizable: true
        },
        permissions: app.permissions || ['filesystem.read.*',
'filesystem.write.user.*'],
        desktopIcon: true // Explicitly set desktop icon to true
    };

    // Add new app
    installedApps.push(appToInstall);

    // Save to local storage
    localStorage.setItem('gos_installed_apps',
JSON.stringify(installedApps));
```

```
// Create app manifest file in the filesystem if possible
try {
    const username = this.kernel.currentUser.username;
    const appManifestPath = `/users/${username}/Apps/${app.id}.json`;

    // Create directory if it doesn't exist
    if
(!this.kernel.modules.filesystem.directoryExists(`/users/${username}/Apps`)) {

this.kernel.modules.filesystem.createDirectory(`/users/${username}/Apps`);
}

        // Save app manifest
        this.kernel.modules.filesystem.writeFile(appManifestPath,
JSON.stringify(appToInstall, null, 2));
    } catch (e) {
        this.log.warn(`Could not save app manifest to filesystem:
${e.message}`);
        // Continue anyway as we've saved to localStorage
    }

        // Show notification
        this.kernel.modules.ui.showNotification('Success', `${app.title}
installed successfully`, 3000);

        // Refresh installed apps tab
        this._loadInstalledApps(this.kernel.modules.process.windowId);

        // Refresh desktop icons to show new app
        try {
            this.kernel._createDesktopIcons();
        } catch (e) {
            this.log.warn(`Could not refresh desktop icons: ${e.message}`);
        }
    }

/**
 * Uninstall an app
 * @private
 * @param {string} appId - App ID
 * @param {string} windowId - Window ID
 */
_uninstallApp(appId, windowId) {
    // Get installed apps from local storage
    const installedApps =
JSON.parse(localStorage.getItem('gos_installed_apps') || '[]');

    // Find app
    const appIndex = installedApps.findIndex(app => app.id === appId);

    if (appIndex === -1) {
        this.kernel.modules.ui.showNotification('Error', 'App not found',
3000);
```

```
        return;
    }

    // Get app name for notification
    const appTitle = installedApps[appIndex].title;

    // Remove app
    installedApps.splice(appIndex, 1);

    // Save to local storage
    localStorage.setItem('gos_installed_apps',
JSON.stringify(installedApps));

    // Remove app manifest from filesystem if possible
    try {
        const username = this.kernel.currentUser.username;
        const appManifestPath = `/users/${username}/Apps/${appId}.json`;

        // Delete file if it exists
        if (this.kernel.modules.filesystem.fileExists(appManifestPath)) {
            this.kernel.modules.filesystem.deleteFile(appManifestPath);
        }
    } catch (e) {
        this.log.warn(`Could not remove app manifest from filesystem: ${e.message}`);
        // Continue anyway as we've removed from localStorage
    }

    // Show notification
    this.kernel.modules.ui.showNotification('Success', `${appTitle} uninstalled successfully`, 3000);

    // Refresh installed apps tab
    this._loadInstalledApps(windowId);

    // Refresh desktop icons to remove app
    try {
        this.kernel._createDesktopIcons();
    } catch (e) {
        this.log.warn(`Could not refresh desktop icons: ${e.message}`);
    }
}

/**
 * Launch an app
 * @private
 * @param {string} appId - App ID
 */
_launchApp(appId) {
    // Get installed apps from local storage
    const installedApps =
JSON.parse(localStorage.getItem('gos_installed_apps') || '[]');

    // Find app
```

```
const app = installedApps.find(app => app.id === appId);

if (!app) {
    this.kernel.modules.ui.showNotification('Error', 'App not found',
3000);
    return;
}

// Use the kernel's application launching mechanism
try {
    this.kernel.launchApplication(appId);
} catch (error) {
    this.kernel.modules.ui.showNotification('Error', `Failed to launch
${app.title}: ${error.message}`, 5000);
}
}

/**
 * Check if an app is installed
 * @private
 * @param {string} appId - App ID
 * @returns {boolean} True if app is installed
 */
_isAppInstalled(appId) {
    // Get installed apps from local storage
    const installedApps =
JSON.parse(localStorage.getItem('gos_installed_apps') || '[]');

    // Check if app is installed
    return installedApps.some(app => app.id === appId);
}

/**
 * Get app data from form
 * @private
 * @param {string} windowId - Window ID
 * @returns {Object} App data
 */
_getAppDataFromForm(windowId) {
    // Get form values
    const appId = document.getElementById(`#${windowId}-app-id`).value;
    const appTitle = document.getElementById(`#${windowId}-app-
title`).value;
    const appIcon = document.getElementById(`#${windowId}-app-icon`).value;
    const appDescription = document.getElementById(`#${windowId}-app-
description`).value;
    const appAuthor = document.getElementById(`#${windowId}-app-
author`).value;
    const appVersion = document.getElementById(`#${windowId}-app-
version`).value;
    const appCategory = document.getElementById(`#${windowId}-app-
category`).value;
    const appCode = document.getElementById(`#${windowId}-app-code`).value;
```

```
// Get permissions
const permissions = [];
if (document.getElementById(`#${windowId}-perm-filesystem`).checked) {
    permissions.push('filesystem');
}
if (document.getElementById(`#${windowId}-perm-network`).checked) {
    permissions.push('network');
}
if (document.getElementById(`#${windowId}-perm-system`).checked) {
    permissions.push('system');
}

// Create app data
return {
    id: appId,
    title: appTitle,
    icon: appIcon,
    description: appDescription,
    author: appAuthor,
    version: appVersion,
    category: appCategory,
    permissions: permissions,
    code: appCode
};

/***
 * Save an app
 * @private
 * @param {Object} appData - App data
 */
_saveApp(appData) {
    // Get developer apps from local storage
    const developerApps =
JSON.parse(localStorage.getItem('gos_developer_apps') || '[]');

    // Check if app already exists
    const appIndex = developerApps.findIndex(app => app.id ===
appData.id);

    if (appIndex !== -1) {
        // Update app
        developerApps[appIndex] = appData;
    } else {
        // Add new app
        developerApps.push(appData);
    }

    // Save to local storage
    localStorage.setItem('gos_developer_apps',
JSON.stringify(developerApps));
}

/***
```

```
* Initialize Diagnostics application (admin only)
* @private
* @param {HTMLElement} container - Container element
* @param {Object} process - Process object
* @param {Object} appInfo - Application metadata
*/
async _initializeDiagnostics(container, process, appInfo) {
    container.innerHTML = `
        <div style="padding:20px; font-size:14px;">
            <button id="${process.windowId}-run-tests"
class="button button-primary">Run Self Tests</button>
            <button id="${process.windowId}-toggle-console"
class="button" style="margin-left:10px;">Toggle Debug Console</button>
            <pre id="${process.windowId}-test-output"
style="margin-top:15px;height:200px;overflow:auto;background:var(--terminal-
bg);padding:10px;"></pre>
        </div>`;

    const runBtn = document.getElementById(`#${process.windowId}-run-
tests`);
    const out = document.getElementById(`#${process.windowId}-test-
output`);
    runBtn.addEventListener('click', async () => {
        out.textContent = 'Running tests...';
        try {
            const result = await
this.kernel.services.get('system').call('runSelfTests');
            const lines = Object.entries(result).map(([k,v]) =>
`#${k}: ${v ? 'OK' : 'FAIL'}`);
            out.textContent = lines.join('\n');
        } catch (e) {
            out.textContent = 'Error: ' + e.message;
        }
    });
}

const toggleBtn = document.getElementById(`#${process.windowId}-toggle-
console`);
if (toggleBtn) {
    toggleBtn.addEventListener('click', () => {
        this.kernel.modules.ui.toggleDebugConsole();
    });
}

async _renderUserManagement(windowId) {
    const container = document.getElementById('user-list');
    try {
        const users = await this.kernel.api.users.call('listUsers');
        let userListHtml = '';
        users.forEach(user => {
            userListHtml += `
                <div class="user-list-item" style="display: flex; justify-
content: space-between; align-items: center; padding: 10px; border-bottom: 1px
solid var(--main-border);">

```

```
<div>
    <strong>${user.name}</strong> (${user.username})
<br>
    <small>Roles: ${user.roles.join(', ')})</small>
</div>
<div>
    <button class="button" data-action="edit" data-
username="${user.username}">Edit</button>
    ${user.username !== 'admin' ? `<button
class="button" data-action="delete" data-
username="${user.username}">Delete</button>` : ''}
    </div>
</div>
`;
});

container.innerHTML = userListHtml;

// Add event listeners for edit/delete buttons
container.querySelectorAll('button').forEach(btn => {
    btn.onclick = async (e) => {
        const action = e.target.dataset.action;
        const username = e.target.dataset.username;

        if (action === 'delete') {
            this.kernel.modules.ui.showDialog('Confirm Deletion',
`Are you sure you want to delete user '${username}'? This cannot be undone.`,
{
buttons: ['Cancel', 'Delete']
})
            .then(async (result) => {
                if (result.button === 'Delete') {
                    try {
                        await
this.kernel.api.users.call('deleteUser', { username });

this.kernel.modules.ui.showNotification('Success', `User '${username}' has been
deleted.`);
                        this._renderUserManagement(windowId);
                    }
                    catch (error) {

this.kernel.modules.ui.showNotification('Error', error.message, 5000);
                }
            }
        });
    }

    if (action === 'edit') {
        try {
            const user = await
this.kernel.api.users.call('getUserDetails', { username });

// ... code to build rolesCheckboxes and
dialogBody remains the same ...
            const allRoles = ['user', 'developer', 'admin'];
            const rolesCheckboxes = allRoles.map(role => `
```

```
        <label style="margin-right: 15px; display: inline-block;">
            <input type="checkbox" name="roles" value="${role}" ${user.roles.includes(role) ? 'checked' : ''}>
            ${role.charAt(0).toUpperCase() + role.slice(1)}
        </label>
    `).join('');

    const dialogBody = `
        <div class="form-group" style="margin-bottom: 15px;">
            <label for="edit-name" style="display: block; margin-bottom: 5px;">Full Name</label>
            <input type="text" id="edit-name" value="${user.name}" style="width: 100%; padding: 8px;">
        </div>
        <div class="form-group" style="margin-bottom: 15px;">
            <label style="display: block; margin-bottom: 5px;">Roles</label>
            <div id="roles-container">${rolesCheckboxes}</div>
        </div>
        <hr style="margin: 20px 0; border-color: var(--main-border); opacity: 0.5;">
        <button class="button" id="change-password-dialog-btn">Change Password</button>
    `;

    // This part remains the same
    this.kernel.modules.ui.showDialog(`Edit User: ${username}`, dialogBody, {
        buttons: ['Cancel', 'Save Changes'],
        getFormData: () => {
            const name =
document.getElementById('edit-name')?.value;
            const roles =
Array.from(document.querySelectorAll('#roles-container input:checked')).map(cb =>
cb.value);
            return { name, roles };
        }
    }).then(async (result) => {
        if (result.button === 'Save Changes' && result.formData) {
            await
this.kernel.api.users.call('updateUser', {
                username,
                name: result.formData.name,
                roles: result.formData.roles
            });
        }
    });

    this.kernel.modules.ui.showNotification('Success', `User '${username}' updated.`);
    this._renderUserManagement(windowId);
}
```

```
        }

    });

    setTimeout(() => {
        const changePasswordBtn =
document.getElementById('change-password-dialog-btn');
        if(changePasswordBtn) {
            changePasswordBtn.onclick = async () => {

                // ✅ FIXED: The showDialog call for
the password now includes formData.

                const passwordResult = await
this.kernel.modules.ui.showDialog('Set New Password', `

                    <div class="form-group">
                        <label for="new-password"
style="display: block; margin-bottom: 5px;">New Password</label>
                        <input type="password"
id="new-password" style="width: 100%; padding: 8px;" autocomplete="new-password">
                    </div>
                `, {
                    buttons: ['Cancel', 'Set
Password'],
                    getFormData: () => {
                        const input =
document.getElementById('new-password');
                        return { password: input ?
input.value : null };
                    }
                });

                // ✅ FIXED: The newPassword value is
now safely retrieved from the dialog's result object.

                if (passwordResult.button === 'Set
Password' && passwordResult.formData) {
                    const newPassword =
passwordResult.formData.password;
                    try {
                        await
this.kernel.api.users.call('setPassword', { username, newPassword });
                    }

                    this.kernel.modules.ui.showNotification('Success', `Password for '${username}' has
been changed.`);
                } catch (error) {

                    this.kernel.modules.ui.showNotification('Error', error.message, 5000);
                }
            };
        }
    }, 100);

} catch (error) {
    this.kernel.modules.ui.showNotification('Error',
error.message, 5000);
}
```

```
        }
    }
};

} catch (e) {
    container.innerHTML = `<p style="color:red;">Error loading users:  
${e.message}</p>`;
}
}

/**  
 * UI Manager  
 * Manages user interface components  
 */  
class UIManager {  
    /**  
     * Create a new UI manager  
     * @param {Kernel} kernel - Kernel reference  
     */  
    constructor(kernel) {  
        this.kernel = kernel;  
        this.log = kernel._createLogger('ui');

        // UI state
        this.currentTheme = 'vintage';
        this.currentFontSize = 'medium';
        this.animations = true;
        this.crtEffects = false;

        this.log.info('UI manager initialized');
    }

    /**
     * Initialize UI manager
     * @async
     */
    async initialize() {
        // Load preferences from localStorage if available
        this._loadPreferences();

        // Apply theme
        this.applyTheme(this.currentTheme);

        // Apply font size
        this.applyFontSize(this.currentFontSize);

        // Apply animation setting
        this.setAnimations(this.animations);

        // Apply CRT effects
        this.setCrtEffects(this.crtEffects);
    }
}
```

```
        return true;
    }

/**
 * Load UI preferences from localStorage
 * @private
 */
_loadPreferences() {
    try {
        const preferences = localStorage.getItem('gos_ui_preferences');
        if (preferences) {
            const prefs = JSON.parse(preferences);
            this.currentTheme = prefs.theme || 'vintage';
            this.currentFontSize = prefs.fontSize || 'medium';
            this.animations = prefs.animations !== false;
            this.crtEffects = prefs.crtEffects === true;
        }
    } catch (error) {
        this.log.error('Failed to load UI preferences:', error);
    }
}

/**
 * Save UI preferences to localStorage
 * @private
 */
_savePreferences() {
    try {
        const preferences = {
            theme: this.currentTheme,
            fontSize: this.currentFontSize,
            animations: this.animations,
            crtEffects: this.crtEffects
        };

        localStorage.setItem('gos_ui_preferences',
JSON.stringify(preferences));
    } catch (error) {
        this.log.error('Failed to save UI preferences:', error);
    }
}

/**
 * Apply theme
 * @param {string} theme - Theme name
 */
applyTheme(theme) {
    // Update body class
    document.body.classList.remove('theme-vintage', 'theme-dark', 'theme-light', 'theme-blue');
    document.body.classList.add(`theme-${theme}`);

    // Store current theme
    this.currentTheme = theme;
}
```

```
// Save preferences
this._savePreferences();

this.log.info(`Applied theme: ${theme}`);
}

/**
 * Apply font size
 * @param {string} size - Font size (small, medium, large)
 */
applyFontSize(size) {
    const sizes = {
        small: '14px',
        medium: '16px',
        large: '18px'
    };

    document.documentElement.style.fontSize = sizes[size] || sizes.medium;

    // Store current font size
    this.currentFontSize = size;

    // Save preferences
    this._savePreferences();

    this.log.info(`Applied font size: ${size}`);
}

/**
 * Set animations enabled/disabled
 * @param {boolean} enabled - Whether animations are enabled
 */
setAnimations(enabled) {
    // Store setting
    this.animations = enabled;

    // Apply setting
    document.documentElement.style.setProperty('--animation-duration',
enabled ? '0.3s' : '0s');

    // Save preferences
    this._savePreferences();

    this.log.info(`Animations ${enabled ? 'enabled' : 'disabled'}`);
}

/**
 * Set CRT effects enabled/disabled
 * @param {boolean} enabled - Whether CRT effects are enabled
 */
setCrtEffects(enabled) {
    // Store setting
    this.crtEffects = enabled;
```

```
// Apply setting
document.getElementById('crt-overlay').style.display = enabled ?
'block' : 'none';
    document.getElementById('scanline').style.display = enabled ? 'block'
: 'none';

// Save preferences
this._savePreferences();

        this.log.info(`CRT effects ${enabled ? 'enabled' :
'disabled'} `);
    }

/** Apply UI translations to static elements */
applyTranslations() {
    const t = (key) => this.kernel.translate(key);
    const heading = document.querySelector('.login-heading');
    const username = document.getElementById('username');
    const password = document.getElementById('password');
    const button = document.querySelector('#login-form button');
    if (heading) heading.textContent = t('login_heading');
    if (username) username.placeholder = t('username');
    if (password) password.placeholder = t('password');
    if (button) button.textContent = t('login_button');
}

/**
 * Show a notification
 * @param {string} title - Notification title
 * @param {string} message - Notification message
 * @param {number} [duration=3000] - Display duration in milliseconds
 */
showNotification(title, message, duration = 3000) {
    const container = document.getElementById('notification-container');

    // Create notification element
    const notification = document.createElement('div');
    notification.className = 'notification';
    notification.innerHTML =
        <div class="notification-title">${title}</div>
        <div class="notification-body">${message}</div>
    `;

    // Add to container
    container.appendChild(notification);

    // Show with animation
    setTimeout(() => {
        notification.classList.add('show');
    }, 10);

    // Remove after duration
    setTimeout(() => {

```

```
        notification.classList.remove('show');

        // Remove from DOM after animation completes
        setTimeout(() => {
            if (notification.parentNode) {
                notification.remove();
            }
        }, 300);
    }, duration);

    return notification;
}

/**
 * Display a persistent error banner
 * @param {string} message - Error message
 */
showErrorBanner(message) {
    const banner = document.getElementById('error-banner');
    if (!banner) return;
    banner.textContent = message;
    banner.style.display = 'block';
}

/** Hide the persistent error banner */
hideErrorBanner() {
    const banner = document.getElementById('error-banner');
    if (banner) banner.style.display = 'none';
}

/** Toggle debug console visibility */
toggleDebugConsole() {
    const consoleEl = document.getElementById('debug-console');
    if (!consoleEl) return;
    if (consoleEl.style.display === 'block') {
        consoleEl.style.display = 'none';
    } else {
        consoleEl.style.display = 'block';
    }
}

/**
 * Display a help dialog with custom content
 * @param {string} html - Help HTML content
 */
showHelp(html) {
    const title = 'Help';
    this.showDialog(title, html, { buttons: ['Close'] });
}

/**
 * Show a dialog, now with complete logic and form data handling.
 * @param {string} title - Dialog title
 * @param {string} message - Dialog message (HTML content)
```

```
* @param {Object} [options] - Dialog options
* @returns {Promise<Object>} Resolves with an object containing the
clicked button and any form data
*/
showDialog(title, message, options = {}) {
    return new Promise(resolve => {
        // Default options from the original file, ensuring full
compatibility
        const defaultOptions = {
            buttons: ['OK'],
            defaultButton: 'OK',
            cancelButton: null,
            getFormData: null // The new callback for capturing form data
        };

        const dialogOptions = { ...defaultOptions, ...options };

        // Create backdrop and modal elements
        const backdrop = document.createElement('div');
        backdrop.className = 'modal-backdrop';
        const dialog = document.createElement('div');
        dialog.className = 'modal';

        // Build dialog content, including the button mapping from the
original file
        dialog.innerHTML =
            `<div class="modal-header">
                <div class="modal-title">${title}</div>
            </div>
            <div class="modal-body">
                ${message}
            </div>
            <div class="modal-footer">
                ${dialogOptions.buttons.map(buttonText => `
                    <button class="button ${buttonText ===
dialogOptions.defaultButton ? 'button-primary' : ''}" data-button="${buttonText}">
                        ${buttonText}
                    </button>
                `).join('')}
            </div>
        `;

        backdrop.appendChild(dialog);
        document.body.appendChild(backdrop);

        // Show dialog with animation
        setTimeout(() => {
            backdrop.classList.add('show');
        }, 10);

        // Function to close the dialog and resolve the promise
        const closeDialog = (buttonName) => {
            let formData = null;
            // ✅ FIXED: Capture form data *before* the modal is removed
```

from the DOM.

```
        if (typeof dialogOptions.getFormData === 'function') {
            formData = dialogOptions.getFormData();
        }

        backdrop.classList.remove('show');
        setTimeout(() => {
            if (backdrop.parentNode) {
                backdrop.remove();
            }
            // Resolve with the consistent object format
            resolve({ button: buttonName, formData: formData });
        }, 300);
    };

    // Set up button handlers
    dialog.querySelectorAll('.button').forEach(button => {
        button.addEventListener('click', () => {
            const buttonName = button.getAttribute('data-button');
            closeDialog(buttonName);
        });
    });

    // Handle Escape key, preserving original functionality
    if (dialogOptions.cancelButton) {
        const escHandler = (e) => {
            if (e.key === 'Escape') {
                document.removeEventListener('keydown', escHandler);
                closeDialog(dialogOptions.cancelButton);
            }
        };
        document.addEventListener('keydown', escHandler);
    }
});

/***
 * Displays the developer registration modal dialog.
 * This method builds the registration form, handles submission,
 * calls the authentication API, and provides user feedback.
 */
showRegistrationDialog() {
    const body = `
        <div class="form-group"><label for="reg-username">Username</label>
<input type="text" id="reg-username" required></div>
        <div class="form-group"><label for="reg-name">Full Name</label>
<input type="text" id="reg-name" required></div>
        <div class="form-group"><label for="reg-password">Password (min. 8
characters)</label><input type="password" id="reg-password" required></div>
        <div id="reg-error" class="login-error"></div>`;

    // This function is passed to showDialog and runs *before* the modal
    // closes,
    // safely capturing the form data.
}
```

```
const getFormData = () => {
    return {
        username: document.getElementById('reg-username').value,
        name: document.getElementById('reg-name').value,
        password: document.getElementById('reg-password').value
    };
};

this.showDialog('Developer Registration', body, { buttons: ['Cancel', 'Register'], getFormData: getFormData })
    .then(async (result) =>
        // Check which button was clicked and if we have form data.
        if (result.button === 'Register' && result.formData) {
            try {
                const apiResult = await
this.kernel.api.auth.call('register', result.formData);
                this.showNotification('Success', apiResult.message);
            } catch (error) {
                // ✅ FIXED: Simply show the server's validation
error in a notification.
                // This is a cleaner and more reliable way to give
user feedback.
                this.showNotification('Registration Failed',
error.message, 5000);
            }
        }
    );
}

}

// Initialize Text Reversion
document.addEventListener('DOMContentLoaded', async () => {
    const debugEl = document.getElementById('debug-console');
    ['log', 'warn', 'error'].forEach(type => {
        const orig = console[type];
        console[type] = function(...args) {
            orig.apply(console, args);
            if (debugEl) {
                debugEl.textContent += `[$${type}] ` + args.join(
') + '\n';
            }
        };
    });
    // Global error handlers
    window.addEventListener('error', e => {
        if (window.kernel && window.kernel.services?.has('system')) {
            window.kernel.services.get('system').call('logError', {
                message: e.message,
                details: e.error?.stack || ''
            }).catch(() => {});
        }
        if (window.kernel && window.kernel.modules?.ui) {
```

```
        window.kernel.modules.ui.showErrorBanner('A system error
occurred: ' + e.message);
    }
});
window.addEventListener('unhandledrejection', e => {
    const msg = e.reason?.message || e.reason;
    const details = e.reason?.stack || '';
    if (window.kernel && window.kernel.services?.has('system')) {
        window.kernel.services.get('system').call('logError', {
            message: msg,
            details: details
        }).catch(() => {});
    }
    if (window.kernel && window.kernel.modules?.ui) {
        window.kernel.modules.ui.showErrorBanner('A system error
occurred: ' + msg);
    }
});

try {
    // Create kernel instance
    window.kernel = new Kernel();

    // Initialize kernel
    await window.kernel.initialize();
} catch (error) {
    console.error('Failed to initialize Text Reversion:', error);

    const splashScreen = document.getElementById('splash-screen');
    if (splashScreen) {
        splashScreen.style.display = 'none';
    }

    const errorScreen = document.getElementById('error-screen');
    if (errorScreen) {
        const errorTitle = document.getElementById('error-title');
        const errorMessage = document.getElementById('error-message');
        const errorDetails = document.getElementById('error-details');

        errorTitle.textContent = 'System Initialization Failed';
        errorMessage.textContent = error.message;

        if (error.stack) {
            errorDetails.textContent = error.stack;
            errorDetails.style.display = 'block';
        }

        errorScreen.style.display = 'flex';
    }
}
});
```

```
/**  
 * ======  
 * GOS Built-in Application Entry Points  
 * ======  
 * This script defines the global functions that the Kernel uses  
 * to launch the built-in applications. Each function serves as  
 * an entry point that initializes the application's UI inside  
 * its designated window.  
 */  
  
/**  
 * Entry point for the Code Editor application.  
 * @param {object} process - The process object from the kernel.  
 */  
function editorApp(process) {  
    const container = document.querySelector(`#${process.windowId} .window-content`);  
    if (container) {  
        // The _initializeCodeEditor method is defined within the Kernel's  
        ProcessManager  
        kernel.modules.process._initializeCodeEditor(container, process, {});  
    }  
}  
  
/**  
 * Entry point for the File Manager application.  
 * @param {object} process - The process object from the kernel.  
 */  
function fileManagerApp(process) {  
    const container = document.querySelector(`#${process.windowId} .window-content`);  
    if (container) {  
        kernel.modules.process._initializeFileManager(container, process, {});  
    }  
}  
  
/**  
 * Entry point for the Mathematical Sandbox application.  
 * @param {object} process - The process object from the kernel.  
 */  
function mathematicalSandboxApp(process) {  
    const container = document.querySelector(`#${process.windowId} .window-content`);  
    if (container) {  
        kernel.modules.process._initializeMathematicalSandbox(container,  
process, {});  
    }  
}  
  
/**  
 * Entry point for the Terminal application.  
 * @param {object} process - The process object from the kernel.  
 */  
function terminalApp(process) {
```

```
    const container = document.querySelector(`#${process.windowId} .window-content`);
    if (container) {
        kernel.modules.process._initializeTerminal(container, process, {});
    }
}

/***
 * Entry point for the Settings application.
 * @param {object} process - The process object from the kernel.
 */
function settingsApp(process) {
    const container = document.querySelector(`#${process.windowId} .window-content`);
    if (container) {
        kernel.modules.process._initializeSettings(container, process, {});
    }
}

/***
 * Entry point for the App Store application.
 * @param {object} process - The process object from the kernel.
 */
function appStoreApp(process) {
    const container = document.querySelector(`#${process.windowId} .window-content`);
    if (container) {
        kernel.modules.process._initializeAppStore(container, process, {});
    }
}

/***
 * Entry point for the Diagnostics application.
 * @param {object} process - The process object from the kernel.
 */
function diagnosticsApp(process) {
    const container = document.querySelector(`#${process.windowId} .window-content`);
    if (container) {
        kernel.modules.process._initializeDiagnostics(container, process, {});
    }
}

/***
 * Entry point for the Developer Center application.
 * @param {object} process - The process object from the kernel.
 */
function devCenterApp(process) {
    const container = document.querySelector(`#${process.windowId} .window-content`);
    if (container) {
        kernel.modules.process._initializeDeveloperCenter(container, process,
{})};
}
```

```
        }

    </script>
</body>
</html>
```

### [34] GenesisOS/lattice-gamev4.js

- **Bytes:** 35135
- **Type:** text

```
// lattice-gamev4.js
// Genesis OS app version of the "x00004 Lattice Game Workbench"
// modelled after clix.js structure

export function initialize(gosApiProxy) {
    const container = gosApiProxy.window.getContainer();
    const doc = container.ownerDocument;
    const win = doc.defaultView;

    // ----- inject styles (taken from lattice-gamev4.html) -----
    const style = doc.createElement("style");
    style.textContent = `
        * { box-sizing: border-box; }
        #lattice-app {
            margin: 0;
            height: 100%;
            display: flex;
            background: #000;
            color: #eee;
            font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
            min-height: 0;
        }
        #lattice-app #panel {
            display: flex;
            flex-direction: column;
            width: 250px;
            background: #121212;
            border-right: 1px solid #2c2c2c;
            padding: 6px;
            gap: 4px;
            min-height: 0;
        }
        #lattice-app #panel-top {
            flex: 0 0 auto;
        }
        #lattice-app h1 { font-size: 0.95rem; margin: 0; }
        #lattice-app label {
            font-size: 0.7rem;
            color: #ddd;
```

```
        display: block;
        margin-top: 4px;
    }
    #lattice-app input,
    #lattice-app select,
    #lattice-app button,
    #lattice-app textarea {
        width: 100%;
        background: #0f0f0f;
        border: 1px solid #333;
        color: #eee;
        border-radius: 4px;
        padding: 4px 6px;
        font-size: 0.7rem;
    }
    #lattice-app button { cursor: pointer; background: #222; }
    #lattice-app button:hover { background: #2e2e2e; }
    #lattice-app textarea {
        font-family: ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas,
        "Liberation Mono", "Courier New", monospace;
        resize: vertical;
    }
    #lattice-app small { font-size: 0.6rem; color: #aaa; }
    #lattice-app #viewport {
        flex: 1;
        position: relative;
        background: radial-gradient(circle at top, #050505 0%, #000 80%);
        overflow: hidden;
        min-width: 0;
    }
    #lattice-app #canvas { width: 100%; height: 100%; display: block; }
    #lattice-app .legend { font-size: 0.6rem; line-height: 1.4; }
    #lattice-app .legend-row { display: flex; align-items: center; gap: 4px;
margin-bottom: 2px; }
    #lattice-app .swatch { width: 10px; height: 10px; border-radius: 9999px; }
    #lattice-app #coordInfo { font-size: 0.6rem; color: #ccc; }
    #lattice-app #log {
        background: #0f0f0f;
        border: 1px solid #333;
        min-height: 34px;
        max-height: 70px;
        overflow-y: auto;
        font-size: 0.6rem;
        padding: 4px 6px;
        white-space: pre-wrap;
    }
    #lattice-app #meshGallery {
        background: #111;
        border: 1px solid #333;
        max-height: 140px;
        overflow-y: auto;
        font-size: 0.65rem;
        padding: 4px;
    }
```

```
#lattice-app .mesh-item {
    padding: 2px 4px;
    cursor: pointer;
    border-radius: 3px;
}
#lattice-app .mesh-item:hover { background: #2c2c2c; }
#lattice-app .mesh-item.active { background: #444; }
#lattice-app #meshJson {
    min-height: 100px;
    max-height: 140px;
}
#lattice-app #api-doc {
    background: #121212;
    border: 1px solid #2c2c2c;
    border-radius: 6px;
    padding: 6px;
    margin-top: 6px;
    max-height: 300px;
    overflow-y: auto;
}
#lattice-app #api-doc .doc-title {
    font-size: 0.7rem;
    font-weight: 600;
    margin-bottom: 4px;
}
#lattice-app #api-doc details { margin-bottom: 4px; }
#lattice-app #api-doc summary { cursor: pointer; font-size: 0.65rem; }
#lattice-app #api-doc .doc-code {
    background: #000;
    border: 1px solid #333;
    border-radius: 4px;
    padding: 4px 6px;
    font-size: 0.6rem;
    white-space: pre;
    overflow-x: auto;
}
#lattice-app #api-doc .doc-text {
    font-size: 0.6rem;
    line-height: 1.3;
    margin-top: 4px;
}
#lattice-app #api-doc .doc-text h4 {
    font-size: 0.62rem;
    margin: 5px 0 2px;
}
#lattice-app #api-doc .doc-text ul,
#lattice-app #api-doc .doc-text ol {
    padding-left: 16px;
    margin: 3px 0;
}
`;
doc.head.appendChild(style);

// ----- inject HTML -----
```

```
container.innerHTML = `<div id="lattice-app">
  <div id="panel">
    <div id="panel-top">
      <h1>x00004 Lattice Game</h1>
      <label>
        Dimension
        <select id="dimension">
          <option value="1">1D</option>
          <option value="2" selected>2D</option>
          <option value="3">3D</option>
        </select>
      </label>
      <label>
        Extent per dimension
        <input id="extent" type="number" value="6" min="1" max="48" />
      </label>
      <label>
        Basis X (default 2)
        <input id="basisX" type="number" value="2" min="1" />
      </label>
      <label>
        Basis Y (default 3)
        <input id="basisY" type="number" value="3" min="1" />
      </label>
      <label>
        Basis Z (default 5)
        <input id="basisZ" type="number" value="5" min="1" />
      </label>

      <label>
        Rotate Y (3D)
        <input id="rotY" type="range" min="-180" max="180" value="-45" />
      </label>
      <label>
        Rotate X (3D)
        <input id="rotX" type="range" min="-90" max="90" value="25" />
      </label>
      <label>
        Zoom
        <input id="zoom" type="range" min="0.3" max="2.5" value="1.1"
step="0.05" />
      </label>

      <div class="legend">
        <div class="legend-row"><span class="swatch" style="background:#fff;">
</span>valid island (x00004.03)</div>
        <div class="legend-row"><span class="swatch" style="background:#555;">
</span>invalid island (code division)</div>
        <div class="legend-row"><span class="swatch" style="background:#f90;">
</span>even-toggled (02)</div>
      </div>
    </div>
```

```
Game script (JS)
<textarea id="scriptBox" style="min-height:90px">/> Example:
engine.buildMeshes();
engine.showAllMeshes();</textarea>
</label>
<button id="runScript">Run Script</button>
<div id="log"></div>

<label style="margin-top:6px;">Mesh Gallery (auto)</label>
<div id="meshGallery"></div>

<label style="margin-top:6px;">Mesh JSON (auto)</label>
<textarea id="meshJson" readonly></textarea>

<div id="api-doc">
<div class="doc-title">JavaScript Scripting API</div>

<details open>
<summary>engine object</summary>
<pre class="doc-code">
// core
engine.getPoints()
engine.read(i,j,k)
engine.write(i,j,k,value)
engine.toggle(i,j,k)
engine.highlight(i,j,k,color)
engine.clearHighlights()
engine.draw()

// mesh
engine.buildMeshes()
engine.getMeshes()
engine.showMesh(index)
engine.showAllMeshes()

// export (if you added JSON download earlier)
engine.exportMeshesJSON??.()
engine.downloadMeshesJSON??.("mesh-db.json")
</pre>
</details>

<details>
<summary>Data: point</summary>
<pre class="doc-code">
{
  i: Number,      // 0-based lattice index X
  j: Number,      // 0-based lattice index Y (1D → 0)
  k: Number,      // 0-based lattice index Z (1D/2D → 0)
  valid: Bool    // x00004.03 says this is a valid island
}
</pre>
</details>

<details>
```

```
<summary>Data: mesh (from engine.getMeshes())</summary>
<pre class="doc-code">
[
  { i:0, j:0, k:0 },
  { i:1, j:0, k:0 },
  ...
]
</pre>
</details>

<details>
<summary>Recipe: build + view all</summary>
<pre class="doc-code">
engine.buildMeshes();
const meshes = engine.getMeshes();
engine.showAllMeshes();
</pre>
</details>

<details>
<summary>Recipe: step through meshes</summary>
<pre class="doc-code">
engine.buildMeshes();
const ms = engine.getMeshes();
let idx = 0;
function nextMesh() {
  engine.showMesh(idx);
  idx = (idx + 1) % ms.length;
}
nextMesh(); // call again to advance
</pre>
</details>

<details>
<summary>Recipe: use lattice as memory</summary>
<pre class="doc-code">
// write a value (stored in toggle-map space)
engine.write(2,0,0,5);

// read it back
const cell = engine.read(2,0,0);
console.log(cell.toggles, cell.isValid);
</pre>
</details>

<details>
<summary>Render flags (if present)</summary>
<pre class="doc-code">
engine.setRender?.({
  invalid: false, // hide invalid islands
  edges: true, // draw mesh edges
  vertices: true, // draw mesh vertices
  angles: false // show angle hints
});
</pre>
</details>
```

```
</pre>
</details>

<details>
<summary>Mesh JSON schema (export)</summary>
<pre class="doc-code">
{
  "meta": {
    "basis": { "x":2, "y":3, "z":5 },
    "dimension": 3,
    "extent": 6
  },
  "meshes": [
    {
      "id": 0,
      "size": 174,
      "vertices": [ { "i":0,"j":0,"k":0 }, ... ],
      "edges": [ [ {i:0,j:0,k:0}, {i:1,j:0,k:0} ], ... ],
      "degree": { "0,0,0": 3, ... },
      "bounds": { "i": [0,5], "j": [0,5], "k": [0,5] },
      "centroid": { "i": 2.4, "j": 1.6, "k": 0, "X": ..., "Y": ..., "Z": ... },
      "angles": { "0,0,0": [90,90,...], ... }
    }
  ]
}
</pre>
</details>
<details>
<summary>Engineering Science Notes (full)</summary>
<div class="doc-text">


<strong>Short answer:</strong> you just gave yourself a way to <strong>spawn repeatable, addressable, geometry-based experiments</strong> on demand. That turns “I have an idea for a structure / network / controller” into “I can generate a clean discrete space, label the good cells, and run code on it” – in 1D, 2D, or 3D – with the same rules every time.</p>



#### 1. Deterministic = testable engineering experiments



Because the lattice is generated the same way from the same inputs (extent, basis, dimension, x00004.03), you can:



- run Algorithm A on the lattice,
- run Algorithm B on the exact same lattice,
- compare outcomes.



That’s basically “same geometry, different engineering method.” That’s what people try to do in CFD, FEA, control experiments, etc. You’ve got a toy-sized but totally reproducible discrete domain.



#### 2. Valid vs invalid = structural vs control planes



Your “valid islands” and “invalid islands” separation is sneaky powerful:



- valid = where the “thing” lives (structure,

```

```
path, data, terrain)</li>
    <li><strong>invalid</strong> = out-of-band (metadata, code
division, channel separation, keying)</li>
    </ul>
    <p>That's basically a <strong>built-in multiplexing scheme.</strong> In engineering science terms, you've got:</p>
    <ul>
        <li>a <strong>primary field</strong> (the realizable nodes),
and</li>
        <li>a <strong>sideband field</strong> (constraints, priorities,
security tags).</li>
        </ul>
        <p>This lets you prototype:</p>
        <ul>
            <li>routing with guard channels,</li>
            <li>multi-layer additive manufacturing patterns,</li>
            <li>logic-on-a-grid where some cells cannot be used.</li>
        </ul>

<h4>3. 1D / 2D / 3D → three classic engineering problem
classes</h4>
    <p><strong>1D (lines):</strong></p>
    <ul>
        <li>signal / pipeline models</li>
        <li>timing chains</li>
        <li>“instruction buses” like CLIX</li>
        <li>serial inspection: walk from cell 0 → N deterministically</li>
    </ul>
    <p><strong>2D (panels):</strong></p>
    <ul>
        <li>circuit boards / floorplans / cellular automata</li>
        <li>layer-by-layer toolpaths</li>
        <li>image/bitmap → structure mappings</li>
        <li>process scheduling as a grid</li>
    </ul>
    <p><strong>3D (volumes):</strong></p>
    <ul>
        <li>structural frames / trusses / voxel materials</li>
        <li>pathfinding / coverage in space (drones, arms, scanners)</li>
        <li>volumetric encryption or stego (hide data in valid pockets)
    </ul>
</li>
    <li>layout for programmable matter / 3D-printed metamaterials</li>
</ul>
<p>You just unified those under one generator.</p>

<h4>4. A playground for lattice-based crypto / stego</h4>
<p>Now that the meshes are deterministic and exportable as JSON,
you can:</p>
<ol>
    <li>Treat the <strong>mesh shape itself</strong> as the key (only
someone who can regenerate your x00004.03 lattice with the same params can
interpret it).</li>
        <li>Use <strong>invalid</strong> cells as code-division slots.
</li>
```

```
        <li>Play with parity toggles (01, 02) as a  
signaling layer.</li>  
    </ol>  
    <p>This is very close to structure-as-key  
cryptography: the message only makes sense when embedded in the right lattice.</p>  
  
    <h4>5. Procedural CAD / Generative engineering</h4>  
    <p>You now have “procedural geometry that is guaranteed to be  
clean.” That means you can do:</p>  
    <ul>  
        <li><strong>parameter sweeps</strong>: vary basis (2,3,5) → get  
different density lattices</li>  
        <li><strong>component extraction</strong>: each mesh = 1  
component</li>  
        <li><strong>library building</strong>: export JSON → feed to  
another tool → loft/extrude → get printable parts</li>  
    </ul>  
    <p>That’s a mini engineering-PCG pipeline:  
deterministic grid → extract topologies → push to CAD.</p>  
  
    <h4>6. Discrete control / robotics thinking</h4>  
    <p>A 3D lattice with edges is literally a graph with known  
adjacency. That’s the starting point for:</p>  
    <ul>  
        <li>coverage algorithms (visit all valid nodes)</li>  
        <li>shortest path in constrained structure</li>  
        <li>task allocation (assign robots to sub-meshes)</li>  
        <li>safety zones (invalid = don’t go there)</li>  
    </ul>  
    <p>Because you can regenerate the same lattice again, you can test  
controllers on the identical graph – same plant, different controller.  
</p>  
  
    <h4>7. Metamaterials & structural patterning (discrete)</h4>  
    <p>Your generator is making repeatable cell complexes.  
</p> The mesh DB already has:</p>  
    <ul>  
        <li>vertices</li>  
        <li>edges</li>  
        <li>bounds</li>  
        <li>degrees</li>  
    </ul>  
    <p>So you can:</p>  
    <ul>  
        <li>assign stiffness per edge,</li>  
        <li>run a quick constraint check,</li>  
        <li>label load paths,</li>  
        <li>study “remove every 3rd valid cell” in a controlled space.  
</li>  
    </ul>  
  
    <h4>8. Education / formal methods bridge</h4>  
    <p>You can show, on one page:</p>  
    <ol>
```

```
<li>the rule (x00004.03),</li>
<li>the result (the lattice),</li>
<li>the machine-readable form (JSON),</li>
<li>the program that runs on it (the JS textbox).</li>
</ol>
<p>That's the whole <strong>spec → model → artifact</strong> cycle  
– very good for teaching engineering science.</p>
```

#### 9. Integration with CLIX / C700

You already have banks, resolvers, pointer-style instruction definitions. Now you also have a **geometric address space.** So you can say:

```
<blockquote>“Instruction x00004.03 applies to the 3D lattice where  
valid cells form mesh #2.”</blockquote>
<p>That's spatial scoping of instructions – geometry-  
scoped programs.</strong></p>
```

#### TL;DR

```
<ul>
<li><strong>Repeatable experiments</strong> (same geometry every  
time)</li>
```

```
<li><strong>Clean separation</strong> of data vs control via  
valid/invalid</li>
```

```
<li><strong>A graph you can simulate on</strong> (pathfinding,  
coverage, logistics)</li>
```

```
<li><strong>Shape-as-key crypto/stego</strong></li>
```

```
<li><strong>Pipeline to CAD / 3D printing / structural  
analysis</strong></li>
```

```
<li><strong>Didactic rule → structure → program flow</strong></li>
</ul>
```

```
<p>So this page is basically a small, deterministic  
<strong>geometry lab</strong> for engineering science.</p>
```

```
</div>
```

```
</details>
```

```
<details>
```

```
<summary>x00004.00 – x00004.03 (bank specifics)</summary>
```

```
<div class="doc-text">
```

```
<h4>x00004.00 – Prime basis</h4>
```

```
<p>
```

```
<code>0000 A set of prime numbers, serves as a basis for all  
positive and negative integers (Including zero), by repeated addition and  
subtraction.</code>
```

```
</p>
```

```
<p>
```

```
    This declares the arithmetic foundation: pick a finite set of  
primes (in this bank we later see {2,5,3}) and allow repeated ± operations. With  
repetition and sign you can span  $\mathbb{Z}$ . This is the numeric substrate the rest of the  
bank assumes. :contentReference[oaicite:0]{index=0}
```

```
</p>
```

```
<h4>x00004.01 – Odd-toggle rule</h4>
```

```
<p>
```

```
<code>0000 An odd number of toggle actions (between two states)  
yields the final state being equal to the initial state.</code>
```

```
</p>
<p>
    This is your “parity holds” rule: if you toggle 1, 3, 5... times,
    you end where you started. This is intentionally the reverse of the usual UI
    toggle, but it’s consistent with the rest of your system. We use this in the UI to
    decide how a clicked point should look after an odd count.
:contentReference[oaicite:1]{index=1}
    </p>

    <h4>x00004.02 – Even-toggle rule</h4>
    <p>
        <code>0000 An even number of toggle actions (Between two states)
yields the non initial state as the final state.</code>
    </p>
    <p>
        Complement to 01: after 2, 4, 6... toggles, you must show the
opposite of the initial state. Together 01 + 02 define the parity semantics we
baked into the lattice click-handler. :contentReference[oaicite:2]{index=2}
    </p>

    <h4>x00004.03 – Parameterized numeric construction</h4>
    <p>This is the big one. It’s a staged definition that introduces
symbols and then ties them together:</p>
    <ul>
        <li><strong>Variables:</strong> <code>a</code>, <code>b</code>,
<code>p</code>, <code>k</code>, <code>z</code>, <code>epsilon</code>,
<code>L</code>. </li>
        <li><strong>Prime constraint:</strong> <code>p is less than or
equal to 12 and prime</code>. </li>
        <li><strong>Ordering:</strong> <code>a is greater than b</code>
(later text also says “This is true for values for a and b from 1 to 101010100”).</li>
        <li><strong>Summation:</strong> <code>k</code> is defined as “the
summation of (i + 1) from i = a to i = b”. </li>
        <li><strong>Derived value:</strong> <code>z</code> is “the square
root (k / 2a)”. </li>
        <li><strong>Epsilon set:</strong> epsilon ∈ { e1, e2, e3, e4, e5
} where:
            <ul>
                <li>e1 = 1</li>
                <li>e2 = 0</li>
                <li>e3 = 1 - (decimal part of z)</li>
                <li>e4 = (decimal part of z)</li>
                <li>e5 = -1</li>
            </ul>
        </li>
        <li><strong>L-member rule:</strong> <code>L</code> is a member of
{ 2, 5, 3 } – which matches the prime basis introduced in 00.</li>
        <li><strong>XOR condition:</strong> there is an exclusive-or on
<code>(z - epsilon)</code> and <code>(z + epsilon)</code> being a member of
{2,5,3} – i.e. exactly one of those offsets must land in the basis set.</li>
        <li><strong>Domain:</strong> “This is true for values for a and b
from 1 to 101010100”. </li>
    </ul>
```

```
<p>
    Put differently: x00004.03 is a recipe to turn index-like integers
    (a, b) into a structured numeric neighborhood around <code>z</code>, then select
    an allowable epsilon from a 5-element set, then assert membership of an offset in
    {2,5,3}. That's exactly what we used to decide whether a lattice point is a "valid
    island." :contentReference[oaicite:3]{index=3}
</p>
<p>
    The tail lines <code>0033</code>-<code>0041</code> are the
    resolved / compressed restatements (they restitch the earlier tokens into full
    sentences).
</p>
</div>
</details>
</div>

<small>
    01: odd toggles → stay initial<br>
    02: even toggles → non-initial<br>
    03: z/epsilon/L → valid island<br>
    mesh-mode → only vertices/edges
</small>
<div id="coordInfo">Hover: (-)</div>
</div>
</div>
<div id="viewport">
    <canvas id="canvas"></canvas>
</div>
</div>
`;

// ----- JS logic (adapted from lattice-gamev4.html) -----
const root = container.querySelector("#lattice-app");
const canvas = root.querySelector("#canvas");
const ctx = canvas.getContext("2d");

const dimensionSel = root.querySelector("#dimension");
const extentInput = root.querySelector("#extent");
const basisXInput = root.querySelector("#basisX");
const basisYInput = root.querySelector("#basisY");
const basisZInput = root.querySelector("#basisZ");
const rotYInput = root.querySelector("#rotY");
const rotXInput = root.querySelector("#rotX");
const zoomInput = root.querySelector("#zoom");
const scriptBox = root.querySelector("#scriptBox");
const runScriptBtn = root.querySelector("#runScript");
const coordInfo = root.querySelector("#coordInfo");
const logBox = root.querySelector("#log");
const meshGallery = root.querySelector("#meshGallery");
const meshJson = root.querySelector("#meshJson");

// state
let toggles = new Map();
let points = [];
```

```
let extraColors = new Map();
let meshes = [];
const ViewState = { mode: "all", currentMesh: -1 };

function frac(x) { return x - Math.floor(x); }
function almostEq(a, b, tol = 1e-6) { return Math.abs(a - b) < tol; }
function logTime() { return new Date().toLocaleTimeString(); }
function logMsg(msg) {
    listBox.textContent = `[${logTime()}] ${msg}\n` + listBox.textContent;
}

// rule checker (x00004.03)
function isValidX00004_03(i, j, k) {
    let a = i + 1;
    let b = (typeof j === "number") ? j + 1 : a;
    if (a > b) { const tmp = a; a = b; b = tmp; }

    let K = 0;
    for (let t = a; t <= b; t++) K += (t + 1);

    const zVal = Math.sqrt(K / (2 * a));
    const epsSet = [1, 0, 1 - frac(zVal), frac(zVal), -1];
    const Lset = [2, 5, 3];

    for (const eps of epsSet) {
        const zMinus = zVal - eps;
        const zPlus = zVal + eps;
        const minusIn = Lset.some(v => almostEq(zMinus, v));
        const plusIn = Lset.some(v => almostEq(zPlus, v));
        if (minusIn !== plusIn && (minusIn || plusIn)) return true;
    }
    return false;
}

function getPointId(i, j, k) { return `${i},${j},${k}`; }

function resize() {
    canvas.width = canvas.clientWidth;
    canvas.height = canvas.clientHeight;
    draw();
}

function buildLattice() {
    points = [];
    extraColors.clear();
    meshes = [];
    ViewState.mode = "all";
    ViewState.currentMesh = -1;

    const dim = parseInt(dimensionSel.value, 10);
    const extent = parseInt(extentInput.value, 10);
    const bx = parseFloat(basisXInput.value);
    const by = parseFloat(basisYInput.value);
    const bz = parseFloat(basisZInput.value);
}
```

```
for (let i = 0; i < extent; i++) {
  const maxJ = (dim >= 2) ? extent : 1;
  const maxK = (dim >= 3) ? extent : 1;
  for (let j = 0; j < maxJ; j++) {
    for (let k = 0; k < maxK; k++) {
      const valid = isValidX00004_03(i, j, k);
      const X = i * bx;
      const Y = j * by;
      const Z = k * bz;
      points.push({ i, j, k, X, Y, Z, valid });
    }
  }
}
renderMeshGallery();
meshJson.value = "";
}

function projectPoint(p) {
  const w = canvas.width, h = canvas.height;
  const cx = w / 2, cy = h / 2;
  const zoom = parseFloat(zoomInput.value);
  const rx = parseFloat(rotXInput.value) * Math.PI / 180;
  const ry = parseFloat(rotYInput.value) * Math.PI / 180;

  let x = p.X, y = p.Y, z = p.Z;
  const cosY = Math.cos(ry), sinY = Math.sin(ry);
  let x1 = x * cosY - z * sinY;
  let z1 = x * sinY + z * cosY;

  const cosX = Math.cos(rx), sinX = Math.sin(rx);
  let y1 = y * cosX - z1 * sinX;
  let z2 = y * sinX + z1 * cosX;

  const perspective = 400 / (400 + z2);
  const px = cx + x1 * zoom * perspective * 20;
  const py = cy + y1 * zoom * perspective * 20;
  return { x: px, y: py, depth: z2 };
}

function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.fillStyle = "#000";
  ctx.fillRect(0, 0, canvas.width, canvas.height);

  if (viewState.mode === "mesh" && viewState.currentMesh >= 0 &&
  meshes[viewState.currentMesh]) {
    drawMeshMode(meshes[viewState.currentMesh]);
    return;
  }

  const projected = points.map(p => ({ ...p, ...projectPoint(p) }))
    .sort((a, b) => a.depth - b.depth);
```

```
for (const p of projected) {
    const id = getPointId(p.i, p.j, p.k);
    const toggleCount = toggles.get(id) || 0;
    let color = p.valid ? "#ffffff" : "#555555";
    if (toggleCount > 0 && toggleCount % 2 === 0) color = "#f90";
    if (extraColors.has(id)) color = extraColors.get(id);
    const r = p.valid ? 5 : 3.5;
    ctx.beginPath();
    ctx.arc(p.x, p.y, r, 0, Math.PI * 2);
    ctx.fillStyle = color;
    ctx.fill();
}
}

function drawMeshMode(mesh) {
    const set = new Set(mesh.map(m => getPointId(m.i, m.j, m.k)));
    const proj = new Map();
    for (const m of mesh) {
        const full = points.find(p => p.i === m.i && p.j === m.j && p.k === m.k);
        if (!full) continue;
        proj.set(getPointId(m.i, m.j, m.k), { ...full, ...projectPoint(full) });
    }

    ctx.lineWidth = 1;
    ctx.strokeStyle = "#ffffff";
    for (const m of mesh) {
        const mid = getPointId(m.i, m.j, m.k);
        const mp = proj.get(mid);
        if (!mp) continue;
        const neigh = [
            [m.i+1, m.j, m.k],
            [m.i-1, m.j, m.k],
            [m.i, m.j+1, m.k],
            [m.i, m.j-1, m.k],
            [m.i, m.j, m.k+1],
            [m.i, m.j, m.k-1],
        ];
        for (const [ni,nj,nk] of neigh) {
            const nid = getPointId(ni,nj,nk);
            if (set.has(nid)) {
                const np = proj.get(nid);
                if (!np) continue;
                ctx.beginPath();
                ctx.moveTo(mp.x, mp.y);
                ctx.lineTo(np.x, np.y);
                ctx.stroke();
            }
        }
    }
}

for (const m of mesh) {
    const mid = getPointId(m.i, m.j, m.k);
    const mp = proj.get(mid);
    if (!mp) continue;
```

```
ctx.beginPath();
ctx.arc(mp.x, mp.y, 5, 0, Math.PI * 2);
ctx.fillStyle = "#ffffff";
ctx.fill();
}

}

function buildMeshesDeterministic() {
  const dim = parseInt(dimensionSel.value, 10);
  const extent = parseInt(extentInput.value, 10);
  const pointMap = new Map();
  for (const p of points) pointMap.set(getPointId(p.i, p.j, p.k), p);

  const visited = new Set();
  const outMeshes = [];

  for (let i = 0; i < extent; i++) {
    const maxJ = (dim >= 2) ? extent : 1;
    const maxK = (dim >= 3) ? extent : 1;
    for (let j = 0; j < maxJ; j++) {
      for (let k = 0; k < maxK; k++) {
        const id = getPointId(i,j,k);
        const p = pointMap.get(id);
        if (!p || !p.valid || visited.has(id)) continue;

        const queue = [p];
        visited.add(id);
        const mesh = [];

        while (queue.length) {
          const cur = queue.shift();
          mesh.push({ i: cur.i, j: cur.j, k: cur.k });

          const neigh = [
            [cur.i+1, cur.j, cur.k],
            [cur.i-1, cur.j, cur.k],
            [cur.i, cur.j+1, cur.k],
            [cur.i, cur.j-1, cur.k],
            [cur.i, cur.j, cur.k+1],
            [cur.i, cur.j, cur.k-1],
          ];
          for (const [ni,nj,nk] of neigh) {
            if (ni < 0 || nj < 0 || nk < 0) continue;
            if (ni >= extent || nj >= ((dim>=2)?extent:1) || nk >= ((dim>=3)?extent:1)) continue;
            const nid = getPointId(ni,nj,nk);
            if (visited.has(nid)) continue;
            const np = pointMap.get(nid);
            if (!np || !np.valid) continue;
            visited.add(nid);
            queue.push(np);
          }
        }
      }
    }
  }
}
```

```
        outMeshes.push(mesh);
    }
}
}
meshes = outMeshes;
renderMeshGallery();
renderMeshJson();
}

function renderMeshGallery() {
    meshGallery.innerHTML = "";
    meshes.forEach((mesh, idx) => {
        const div = doc.createElement("div");
        div.className = "mesh-item" + (viewState.currentMesh === idx ? " active" : "");
        div.textContent = `Mesh ${idx} - ${mesh.length} pts`;
        div.addEventListener("click", () => {
            engine.showMesh(idx);
        });
        meshGallery.appendChild(div);
    });
}

function renderMeshJson() {
    const db = meshes.map((m, i) => ({
        id: i,
        size: m.length,
        points: m
    }));
    meshJson.value = JSON.stringify(db, null, 2);
}

// engine API exposed to scripts
const engine = {
    getPoints() {
        return points.map(p => ({ i:p.i, j:p.j, k:p.k, valid:p.valid }));
    },
    toggle(i, j=0, k=0) {
        const id = getPointId(i,j,k);
        const prev = toggles.get(id) || 0;
        toggles.set(id, prev+1);
    },
    highlight(i, j=0, k=0, color="#ffff") {
        const id = getPointId(i,j,k);
        extraColors.set(id, color);
    },
    clearHighlights() { extraColors.clear(); },
    draw() { draw(); },
    read(i, j=0, k=0) {
        const id = getPointId(i,j,k);
        return { toggles: toggles.get(id)||0, isValid: isValidX00004_03(i,j,k) };
    },
    write(i, j=0, k=0, value=1) {
        const id = getPointId(i,j,k);
    }
}
```

```
    toggles.set(id, value);
},
buildMeshes() {
  buildMeshesDeterministic();
  logMsg("Meshes built: " + meshes.length);
  return meshes;
},
getMeshes() { return meshes; },
showMesh(idx) {
  if (!meshes[idx]) return;
  ViewState.mode = "mesh";
  ViewState.currentMesh = idx;
  Array.from(meshGallery.children).forEach((el,i) => {
    el.classList.toggle("active", i === idx);
  });
  draw();
},
showAllMeshes() {
  ViewState.mode = "all";
  ViewState.currentMesh = -1;
  this.clearHighlights();
  const palette = ["#ffffff", "#ff5e5e", "#7df76b", "#5ad1ff", "#ffdb5a",
"#ff9ef5"];
  meshes.forEach((mesh, i) => {
    const col = palette[i % palette.length];
    mesh.forEach(p => this.highlight(p.i,p.j,p.k,col));
  });
  draw();
}
};

// expose to console (optional)
win.engine = engine;

// events
runScriptBtn.addEventListener("click", () => {
try {
  const code = scriptBox.value;
  const fn = new Function("engine", code);
  fn(engine);
  logMsg("Script OK");
} catch (err) {
  logMsg("Script error: " + err.message);
  console.error(err);
}
});

[dimensionSel, extentInput, basisXInput, basisYInput, basisZInput].forEach(el =>
{
  el.addEventListener("input", () => {
    buildLattice();
    draw();
  });
});
```

```
[rotXInput, rotYInput, zoomInput].forEach(el => {
  el.addEventListener("input", draw);
});

// canvas hover info
canvas.addEventListener("mousemove", (e) => {
  const rect = canvas.getBoundingClientRect();
  const x = e.clientX - rect.left;
  const y = e.clientY - rect.top;
  // cheap nearest
  let best = null, bestD = 9999;
  for (const p of points) {
    const pr = projectPoint(p);
    const dx = pr.x - x, dy = pr.y - y;
    const d2 = dx*dx + dy*dy;
    if (d2 < 100 && d2 < bestD) {
      bestD = d2;
      best = p;
    }
  }
  if (best) {
    coordInfo.textContent = `Hover: (${best.i},${best.j},${best.k})`;
    valid=${best.valid};
  } else {
    coordInfo.textContent = "Hover: (-)";
  }
});

canvas.addEventListener("click", (e) => {
  const rect = canvas.getBoundingClientRect();
  const x = e.clientX - rect.left;
  const y = e.clientY - rect.top;
  let best = null, bestD = 9999;
  for (const p of points) {
    const pr = projectPoint(p);
    const dx = pr.x - x, dy = pr.y - y;
    const d2 = dx*dx + dy*dy;
    if (d2 < 100 && d2 < bestD) {
      bestD = d2;
      best = p;
    }
  }
  if (best) {
    const id = getPointId(best.i,best.j,best.k);
    const prev = toggles.get(id) || 0;
    toggles.set(id, prev+1);
    draw();
  }
});

// init
resize();
buildLattice();
draw();
```

```
// resize on window/container resize
win.addEventListener("resize", resize);
}
```

### [35] GenesisOS/lattice-gamev4.json

- **Bytes:** 517
- **Type:** text

```
{
  "id": "lattice-gamev4",
  "title": "x00004 Lattice Game Workbench",
  "description": "Deterministic 1D\\2D\\3D lattice generator with mesh extraction, JS scripting, and x00004.03 valid\\invalid islands \u2014 packaged for Genesis OS.",
  "version": "1.0.0",
  "author": "Dominic Alexander Cooper",
  "icon": "\u2b1b",
  "category": "Development",
  "entry": "lattice-gamev4.js",
  "permissions": [],
  "window": {
    "width": 1250,
    "height": 720,
    "resizable": true
  }
}
```

### [36] GenesisOS/make\_docs\_pdf.py

- **Bytes:** 8419
- **Type:** text

```
# Converts selected text-based files into KDP-ready PDFs (8.25" x 11")
# with a left/right gutter column for line numbers (mirrored margins).
#
# Target: KDP 8.25" x 11" (no bleed), ~424 pages.
#
# This version uses *wide* margins for a more spacious layout:
#   - Inside (gutter): 1.0"
#   - Outside (other horizontal edge): 0.875"
#   - Top: 1.0"
#   - Bottom: 1.0"
#
# Mirroring:
#   - Odd pages (right-hand): inside on LEFT, outside on RIGHT
#   - Even pages (left-hand): inside on RIGHT, outside on LEFT
```

```
#  
# Supported file types (case-insensitive):  
#   .css, .cmd, .env, .gitignore, .js, .php, .hpp, .cpp, .md, .py, .txt, .ps1,  
.json, .html  
#  
# Output: ./docpdf/<basename>.pdf  
#  
# Usage:  
#   pip install reportlab  
#   python make_docs_pdf.py  
  
RECURSIVE = False          # set True to scan subfolders  
INCLUDE_DOTFILES = True    # include dotfiles like .gitignore, .env  
  
import os  
import io  
import textwrap  
from typing import List, Tuple  
  
from reportlab.pdfgen import canvas  
from reportlab.pdfbase import pdfmetrics  
  
# -----  
# PAGE + MARGINS (KDP 8.25" x 11", WIDE)  
# -----  
DPI = 72      # PDF points per inch  
  
TRIM_WIDTH_IN = 8.25  
TRIM_HEIGHT_IN = 11.0  
PAGE_SIZE = (TRIM_WIDTH_IN * DPI, TRIM_HEIGHT_IN * DPI)  # 8.25" x 11"  
  
# Wide but KDP-safe margins  
INSIDE_MARGIN_IN = 1.0      # gutter (spine side)  
OUTSIDE_MARGIN_IN = 0.875   # other horizontal edge  
TOP_MARGIN_IN = 1.0  
BOTTOM_MARGIN_IN = 1.0  
  
INSIDE_MARGIN = INSIDE_MARGIN_IN * DPI  
OUTSIDE_MARGIN = OUTSIDE_MARGIN_IN * DPI  
MARGIN_TOP = TOP_MARGIN_IN * DPI  
MARGIN_BOTTOM = BOTTOM_MARGIN_IN * DPI  
  
# Gutter column for line numbers  
GUTTER_WIDTH = 0.6 * DPI    # reserved column width  
GUTTER_GAP = 0.15 * DPI     # gap between gutter and text  
  
# Header/Footer band heights  
HEADER_HEIGHT = 0.4 * DPI  
FOOTER_HEIGHT = 0.4 * DPI  
  
# Typography  
FONT_NAME = "Courier"  # built-in monospaced font  
FONT_SIZE = 10  
LINE_LEADING = 1.2      # line spacing multiplier
```

```
OUTPUT_DIR = "docpdf"

# Supported extensions and special filenames
EXTS = {
    ".css", ".cmd", ".js", ".php", ".hpp", ".cpp", ".md",
    ".py", ".txt", ".ps1", ".json", ".html", "json" # include bare 'json' just in
case
}
SPECIAL_FILES = {".gitignore", ".env"}

def is_supported_file(path: str) -> bool:
    name = os.path.basename(path)
    lower = name.lower()
    # include special dotfiles regardless of extension
    if lower in SPECIAL_FILES and INCLUDE_DOTFILES:
        return True
    # include if extension matches supported list
    _, ext = os.path.splitext(lower)
    if ext in EXTS or (lower.endswith("json") and ".json" not in ext):
        return True
    return False

def iter_candidate_files(root: str = ".") -> List[str]:
    files = []
    if RECURSIVE:
        for dirpath, _, filenames in os.walk(root):
            for fn in filenames:
                full = os.path.join(dirpath, fn)
                if os.path.isfile(full) and is_supported_file(full):
                    files.append(full)
    else:
        for fn in os.listdir(root):
            full = os.path.join(root, fn)
            if os.path.isfile(full) and is_supported_file(full):
                files.append(full)
    return files

def numeric_key(file: str):
    """Numeric-aware sort: if basename (without ext) is an int, sort by int; else
alpha."""
    base = os.path.basename(file)
    stem, _ = os.path.splitext(base)
    try:
        return (0, int(stem))
    except ValueError:
        return (1, base.lower())

def measure_char_width() -> float:
    """Width of one monospaced character in chosen font/size."""
```

```
    return pdfmetrics.stringWidth("M", FONT_NAME, FONT_SIZE)

def wrap_text_line(line: str, max_chars: int) -> List[str]:
    """Soft-wrap a line to max_chars; tabs -> 4 spaces; keep spacing."""
    line = line.replace("\t", "    ").rstrip("\n\r")
    if max_chars <= 0:
        return [line]
    wrapped = textwrap.wrap(
        line,
        width=max_chars,
        break_long_words=True,
        replace_whitespace=False,
        drop_whitespace=False,
    )
    return wrapped if wrapped else [""]

def draw_header_footer(
    c: canvas.Canvas,
    width: float,
    height: float,
    title: str,
    page_num: int,
    margin_left: float,
    margin_right: float,
):
    """Header: title near left margin, page number near right margin.
    Footer: centered credit line."""
    c.setFont(FONT_NAME, 9)

    # Header
    y_header = height - MARGIN_TOP + (HEADER_HEIGHT / 2) - 3
    c.drawString(margin_left, y_header, title[:120])
    c.drawRightString(width - margin_right, y_header, f"Page {page_num}")

    # Footer
    y_footer = MARGIN_BOTTOM / 2
    c.drawCentredString(
        width / 2,
        y_footer,
        "Prompt Engineered by Dominic Alexander Cooper Cert HE (Open) 2025",
    )

def create_pdf_for_textfile(src_path: str):
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    base = os.path.basename(src_path)
    stem, _ = os.path.splitext(base)
    out_path = os.path.join(OUTPUT_DIR, f"{stem}.pdf")

    # Read file as UTF-8 (replace invalid)
    with io.open(src_path, "r", encoding="utf-8", errors="replace") as f:
        lines = f.readlines()
```

```
c = canvas.Canvas(out_path, pagesize=PAGE_SIZE)
width, height = PAGE_SIZE

char_w = measure_char_width()
line_h = FONT_SIZE * LINE_LEADING

# Frame geometry that doesn't depend on left/right swapping
y_top = height - MARGIN_TOP - HEADER_HEIGHT
y_bottom = MARGIN_BOTTOM + FOOTER_HEIGHT

# text frame width is the same on odd/even pages:
frame_width = width - (INSIDE_MARGIN + OUTSIDE_MARGIN + GUTTER_WIDTH +
GUTTER_GAP)
max_chars = int(frame_width // char_w)

# Will be set per page in start_page()
x_text_left = x_text_right = 0.0
x_gutter_left = x_gutter_right = 0.0

c.setFont(FONT_NAME, FONT_SIZE)

def start_page(page_no: int, title: str):
    nonlocal x_text_left, x_text_right, x_gutter_left, x_gutter_right

    is_odd = (page_no % 2 == 1)

    if is_odd:
        # Odd pages: inside on LEFT, outside on RIGHT
        margin_left = INSIDE_MARGIN
        margin_right = OUTSIDE_MARGIN

        x_gutter_left = margin_left
        x_gutter_right = x_gutter_left + GUTTER_WIDTH
        x_text_left = x_gutter_right + GUTTER_GAP
        x_text_right = width - margin_right
    else:
        # Even pages: inside on RIGHT, outside on LEFT
        margin_left = OUTSIDE_MARGIN
        margin_right = INSIDE_MARGIN

        x_text_left = margin_left
        x_gutter_right = width - margin_right
        x_gutter_left = x_gutter_right - GUTTER_WIDTH
        x_text_right = x_gutter_left - GUTTER_GAP

    draw_header_footer(
        c,
        width,
        height,
        title=title,
        page_num=page_no,
        margin_left=margin_left,
        margin_right=margin_right,
```

```
)  
  
    c.setLineWidth(0.5)  
    # vertical line separating gutter from text  
    c.line(x_gutter_right, height - MARGIN_TOP, x_gutter_right, y_bottom)  
  
    return y_top  
  
page_no = 1  
y = start_page(page_no, title=os.path.relpath(src_path))  
page_no += 1  
  
lineno = 1  
for raw in lines:  
    wrapped = wrap_text_line(raw, max_chars=max_chars)  
    for i, seg in enumerate(wrapped):  
        if y - line_h < y_bottom:  
            c.showPage()  
            y = start_page(page_no, title=os.path.relpath(src_path))  
            page_no += 1  
        # gutter number only on first visual segment  
        num_str = f"{lineno:>5}" if i == 0 else ""  
        if num_str:  
            c.drawString(x_gutter_right - 3, y, num_str)  
            c.drawString(x_text_left, y, seg)  
            y -= line_h  
    lineno += 1  
  
c.save()  
return out_path  
  
  
def main():  
    files = iter_candidate_files(".")  
    files.sort(key=numeric_key)  
    if not files:  
        print("No supported files found.")  
        return  
  
    outputs: List[Tuple[str, str]] = []  
    for path in files:  
        try:  
            out = create_pdf_for_textfile(path)  
            outputs.append((path, out))  
        except Exception as e:  
            print(f"Skipped {path}: {e}")  
  
    print("Generated PDFs:")  
    for src, out in outputs:  
        print(f"  {src} -> {out}")  
  
  
if __name__ == "__main__":
```

```
main()
```

### [37] GenesisOS/memory.txt

- **Bytes:** 79
- **Type:** text

```
4696547894075086198331163092979959979597804330157078601375485
```

```
I AM FROM HERE
```

### [38] GenesisOS/quadtrees-visualizer.js

- **Bytes:** 28176
- **Type:** text

```
export function initialize(gosApiProxy) {
    const container = gosApiProxy.window.getContainer();
    const doc = container.ownerDocument; // Use the iframe's document object

    /**
     * Dynamically loads an external script into the sandboxed environment.
     * @param {string} url - The URL of the script to load.
     * @returns {Promise<void>} A promise that resolves when the script is loaded.
     */
    function loadScript(url) {
        return new Promise((resolve, reject) => {
            const script = doc.createElement('script');
            script.src = url;
            script.onload = resolve;
            script.onerror = () => reject(new Error(`Failed to load script: ${url}`));
            doc.head.appendChild(script);
        });
    }

    /**
     *  FIXED: A more robust function to convert rgb() or rgba() color strings
     * to #RRGGBB hex.
     * @param {string} rgbString - The RGB or RGBA color string.
     * @returns {string} The color in hexadecimal format.
     */
    function rgbToHex(rgbString) {
        if (!rgbString || typeof rgbString !== 'string') return '#ffffff';
        if (rgbString.startsWith('#')) return rgbString;

        const match = rgbString.match(/^rgba?\((\d+),(\d+),(\d+)(?:,(\d+\.\d+))?\)$/);
        if (match) {
            const [_, r, g, b, a] = match;
            const hexR = ('0' + (r / 255).toFixed(2)).slice(-2);
            const hexG = ('0' + (g / 255).toFixed(2)).slice(-2);
            const hexB = ('0' + (b / 255).toFixed(2)).slice(-2);
            let hexA = '';
            if (a) hexA = ('0' + (a / 255).toFixed(2)).slice(-2);
            return `#${hexR}${hexG}${hexB}${hexA}`;
        }
    }
}
```

```
        if (!match) return '#ffffff'; // Default for non-matching formats (e.g.,  
'transparent')  
  
        const toHex = c => ('0' + parseInt(c, 10).toString(16)).slice(-2);  
  
        return `#${toHex(match[1])}${toHex(match[2])}${toHex(match[3])}`;  
    }  
  
    /**  
     * The main function to set up and run the application.  
     */  
    async function main() {  
        try {  
            // Load required external libraries. html2canvas is essential for the  
            export feature.  
            await  
loadScript("https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.m  
in.js");  
  
            // --- 1. Inject CSS Styles ---  
            const style = doc.createElement('style');  
            style.textContent = `  
                /* General Layout & Theming */  
                :root {  
                    --primary-color: #4B5320; /* Army Green */  
                    --secondary-color: #BDB76B; /* Dark Khaki */  
                    --background-color: #F5F5DC; /* Beige */  
                    --text-color: #333333; /* Dark Gray */  
                    --font-main: "Fira Code", "Consolas", monospace;  
                }  
                body {  
                    font-family: var(--font-main);  
                    background-color: var(--background-color);  
                    color: var(--text-color);  
                    margin: 0;  
                    padding: 20px;  
                    height: 100%;  
                    box-sizing: border-box;  
                }  
                header { text-align: center; margin-bottom: 20px; color: var(--  
primary-color); }  
                .operation-section { background-color: #fff; padding: 20px;  
border-radius: 8px; box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); max-width: 1200px;  
margin: auto; }  
                .operation-container { display: flex; gap: 20px; flex-wrap: wrap;  
}  
                .operation-controls { flex: 1; min-width: 250px; }  
                .operation-result { flex: 2; min-width: 400px; }  
                h1, h2, h3 { color: var(--primary-color); }  
                h2 { margin-top: 0; border-bottom: 2px solid var(--background-  
color); padding-bottom: 10px; }  
                .form-group { margin-bottom: 15px; }  
                label { display: block; margin-bottom: 5px; font-weight: bold; }  
                input[type="number"] { width: 100%; padding: 8px; border: 1px
```

```
solid #ccc; border-radius: 4px; box-sizing: border-box; }
    .control-row { display: flex; gap: 10px; margin-top: 20px; }
        button { padding: 10px 15px; border: none; border-radius: 4px;
background-color: var(--primary-color); color: white; cursor: pointer; font-
family: var(--font-main); transition: background-color 0.3s; }
            button:hover { background-color: #3a411a; }
            #quadtree-container { position: relative; border: 2px solid var(--secondary-color); background-color: #f9f9f9; margin: auto; }
                .quadtree-cell { position: absolute; box-sizing: border-box;
border: 1px solid #ccc; }
                    .cell-content { display: flex; justify-content: center; align-
items: center; width: 100%; height: 100%; font-size: 12px; color: #555; user-
select: none; }
                        .quadtree-cell.can-subdivide { cursor: pointer; }
                        .quadtree-cell.can-subdivide:hover { background-color: rgba(0, 0,
0, 0.05); }
                        .quadtree-context-menu { position: absolute; background-color:
white; border: 1px solid var(--secondary-color); border-radius: 4px; box-shadow: 0
2px 10px rgba(0, 0, 0, 0.2); padding: 8px 0; min-width: 200px; z-index: 1000;
display: none; }
                            .quadtree-context-menu.active { display: block; }
                            .context-menu-item { padding: 8px 16px; cursor: pointer;
transition: background-color 0.2s ease; display: flex; align-items: center; user-
select: none; }
                                .context-menu-item:hover { background-color: var(--background-
color); }
                                .context-menu-item-icon { margin-right: 10px; width: 16px; text-
align: center; }
                                    .context-menu-separator { height: 1px; background-color: var(--secondary-color);
margin: 6px 0; opacity: 0.3; }
                                    .quadtree-cell-custom-text { position: absolute; top: 50%; left:
50%; transform: translate(-50%, -50%); white-space: nowrap; overflow: hidden;
text-overflow: ellipsis; max-width: 90%; font-family: var(--font-main); pointer-
events: none; }
                                        .quadtree-cell-image { position: absolute; top: 0; left: 0; width:
100%; height: 100%; object-fit: cover; pointer-events: none; }
                                        .quadtree-cell.customized .cell-content { display: none; }
                                        .quadtree-cell.paste-focus { outline: 2px dashed var(--primary-
color); z-index: 10; }
                                            .paste-indicator { position: fixed; bottom: 20px; right: 20px;
background-color: var(--background-color); border: 1px solid var(--secondary-
color); border-left: 4px solid var(--primary-color); padding: 10px 15px; font-
size: 0.9rem; box-shadow: 0 2px 10px rgba(0,0,0,0.1); border-radius: 4px; z-index:
1000; display: none; }
                                                .paste-indicator.active { display: block; animation: fadeIn 0.3s
ease-in-out; }
                                                .export-loading { position: fixed; top: 0; left: 0; width: 100%;
height: 100%; background-color: rgba(0, 0, 0, 0.5); display: none; justify-
content: center; align-items: center; z-index: 2000; }
                                                    .export-loading.active { display: flex; }
                                                    .export-loading-content { background-color: white; padding: 20px;
border-radius: 4px; text-align: center; }
                                                        .export-spinner { border: 4px solid #f3f3f3; border-top: 4px solid
var(--primary-color); border-radius: 50%; width: 40px; height: 40px; animation:
```

```
spin 2s linear infinite; margin: 0 auto 15px auto; }
    @keyframes spin { 0% { transform: rotate(0deg); } 100% {
transform: rotate(360deg); } }
    @keyframes fadeIn { from { opacity: 0; transform:
translateY(10px); } to { opacity: 1; transform: translateY(0); } }
.quadtree-modal { position: fixed; top: 0; left: 0; width: 100%;
height: 100%; background-color: rgba(0, 0, 0, 0.5); display: none; justify-
content: center; align-items: center; z-index: 1100; }
.quadtree-modal.active { display: flex; }
.quadtree-modal-content { background-color: white; padding: 20px;
border-radius: 4px; max-width: 400px; width: 90%; }
.quadtree-modal-header { display: flex; justify-content: space-
between; align-items: center; margin-bottom: 20px; }
.quadtree-modal-header h3 { margin: 0; }
.quadtree-modal-close { background: none; border: none; font-size:
1.5rem; cursor: pointer; padding: 0; color: var(--text-color); }
.quadtree-modal-body { margin-bottom: 20px; }
.quadtree-form-actions { display: flex; justify-content: flex-end;
gap: 10px; }
.button-secondary { background-color: #6c757d; }
.button-secondary:hover { background-color: #5a6268; }
`;
doc.head.appendChild(style);

// --- 2. Inject HTML Structure ---
container.innerHTML =
<header>
    <h1>Quadtree Operations</h1>
</header>
<main>
    <section id="tree-operations" class="operation-section">
        <div class="operation-container">
            <div class="operation-controls">
                <h2>Quadtree Settings</h2>
                <div class="form-group">
                    <label for="max-depth">Max Depth:</label>
                    <input type="number" id="max-depth" min="1"
max="6" value="3">
                </div>
                <div class="form-group">
                    <label for="quadtree-size">Size (px):</label>
                    <input type="number" id="quadtree-size"
min="200" max="800" value="400">
                </div>
                <div class="control-row">
                    <button id="create-quadtreenode">Create</button>
                    <button id="reset-quadtreenode">Reset</button>
                    <button id="export-quadtreenode">Export
PNG</button>
                </div>
            </div>
            <div class="operation-result">
                <h2>Result</h2>
                <div id="quadtree-container"></div>
            </div>
        </div>
    </section>
</main>
```

```
        </div>
    </div>
</section>
</main>
<input type="file" id="quadtree-image-input" accept="image/*"
style="display:none;">
`;

// --- 3. Application Logic ---
const maxDepthInput = doc.getElementById('max-depth');
const quadtreeSizeInput = doc.getElementById('quadtree-size');
const createQuadtreeBtn = doc.getElementById('create-quadtree');
const resetQuadtreeBtn = doc.getElementById('reset-quadtree');
const exportQuadtreeBtn = doc.getElementById('export-quadtree');
const quadtreeContainer = doc.getElementById('quadtree-container');
const imageFileInput = doc.getElementById('quadtree-image-input');

let quadtreeRoot = null;
let maxDepth = 3;
let quadtreeSize = 400;
let activeQuadtreeCell = null;

class QuadtreeNode {
    constructor(x, y, size, depth) {
        this.x = x;
        this.y = y;
        this.size = size;
        this.depth = depth;
        this.children = [];
        this.data = {};
        this.element = this.createElement();
        this.updateVisualState();
    }

    createElement() {
        const cell = doc.createElement('div');
        cell.className = 'quadtree-cell';
        cell.style.left = `${this.x}px`;
        cell.style.top = `${this.y}px`;
        cell.style.width = `${this.size}px`;
        cell.style.height = `${this.size}px`;

        const content = doc.createElement('div');
        content.className = 'cell-content';
        content.textContent = `d: ${this.depth}`;
        cell.appendChild(content);

        cell.addEventListener('click', () => this.handleClick());
        quadtreeContainer.appendChild(cell);
        return cell;
    }

    updateVisualState() {
        if (this.depth < maxDepth) {
```

```
        this.element.classList.add('can-subdivide');
        this.element.title = 'Click to subdivide';
    } else {
        this.element.classList.remove('can-subdivide');
        this.element.title = 'Max depth reached';
    }
    const hue = (this.depth * 30) % 360;
    this.element.style.backgroundColor = `hsl(${hue}, 70%, 90%)`;
    this.element.style.borderColor = `hsl(${hue}, 70%, 60%)`;
}

handleClick() {
    if (this.children.length === 0 && this.depth < maxDepth) {
        this.subdivide();
    }
}

subdivide() {
    if (this.children.length > 0 || this.depth >= maxDepth) return
false;
    const childSize = this.size / 2;
    this.children.push(
        new QuadtreeNode(this.x, this.y, childSize, this.depth +
1),
        new QuadtreeNode(this.x + childSize, this.y, childSize,
this.depth + 1),
        new QuadtreeNode(this.x, this.y + childSize, childSize,
this.depth + 1),
        new QuadtreeNode(this.x + childSize, this.y + childSize,
childSize, this.depth + 1)
    );
    this.element.style.display = 'none';
    return true;
}
}

function initQuadtree() {
    quadtreeContainer.innerHTML = '';
    maxDepth = parseInt(maxDepthInput.value) || 3;
    quadtreeSize = parseInt(quadtreeSizeInput.value) || 400;
    quadtreeContainer.style.width = `${quadtreeSize}px`;
    quadtreeContainer.style.height = `${quadtreeSize}px`;
    quadtreeRoot = new QuadtreeNode(0, 0, quadtreeSize, 0);
}

function initExtensions() {
    quadtreeContainer.addEventListener('contextmenu',
handleQuadtreeContextMenu);
    createContextMenu();
    setupClipboardPaste();
    inputFileInput.addEventListener('change', handleImageUpload);
}

function createContextMenu() {
```

```
if (doc.getElementById('quadtree-context-menu')) return;

const menu = doc.createElement('div');
menu.id = 'quadtree-context-menu';
menu.className = 'quadtree-context-menu';
menu.innerHTML = `
    <div class="context-menu-item" data-action="change-color">
<span class="context-menu-item-icon">⊕</span><span>Change Color...</span></div>
    <div class="context-menu-item" data-action="add-text"><span
class="context-menu-item-icon">T</span><span>Add Text...</span></div>
    <div class="context-menu-item" data-action="add-image"><span
class="context-menu-item-icon">☒</span><span>Add Image...</span></div>
    <div class="context-menu-item" data-action="paste-image"><span
class="context-menu-item-icon">📋</span><span>Paste From Clipboard</span></div>
    <div class="context-menu-separator"></div>
    <div class="context-menu-item" data-action="reset-cell"><span
class="context-menu-item-icon">↺</span><span>Reset Cell</span></div>
`;
doc.body.appendChild(menu);

menu.querySelectorAll('.context-menu-item').forEach(item => {
    item.addEventListener('click', handleContextMenuAction);
});

doc.addEventListener('click', (e) => {
    if (!menu.contains(e.target)) menu.classList.remove('active');
});

const pasteIndicator = doc.createElement('div');
pasteIndicator.id = 'paste-indicator';
pasteIndicator.className = 'paste-indicator';
pasteIndicator.textContent = 'Press Ctrl+V to paste image';
doc.body.appendChild(pasteIndicator);
}

function handleQuadtreeContextMenu(event) {
    event.preventDefault();
    const cell = event.target.closest('.quadtree-cell');
    if (!cell) return;

    activeQuadtreeCell = cell;
    const menu = doc.getElementById('quadtree-context-menu');
    menu.style.left = `${event.clientX}px`;
    menu.style.top = `${event.clientY}px`;
    menu.classList.add('active');
}

function handleContextMenuAction(event) {
    const action = event.currentTarget.getAttribute('data-action');
    const cell = activeQuadtreeCell;
    doc.getElementById('quadtree-context-
menu').classList.remove('active');
    if (!cell) return;
```

```
        switch (action) {
            case 'change-color': showColorPickerModal(cell); break;
            case 'add-text': showTextModal(cell); break;
            case 'add-image': inputFile.click(); break;
            case 'paste-image': prepareCellForPaste(cell); break;
            case 'reset-cell': resetCell(cell); break;
        }
    }

    function markCellAsCustomized(cell) {
        cell.classList.add('customized');
    }

    function showModal(id, content) {
        let modal = doc.getElementById(id);
        if (!modal) {
            modal = doc.createElement('div');
            modal.id = id;
            modal.className = 'quadtree-modal';
            doc.body.appendChild(modal);
        }
        modal.innerHTML = content;
        modal.classList.add('active');
        return modal;
    }

    function showColorPickerModal(cell) {
        // ✅ FIXED: Use the robust converter function here.
        const initialColorRgb = getComputedStyle(cell).backgroundColor;
        const initialColorHex = rgbToHex(initialColorRgb);
        const modalContent = `
            <div class="quadtree-modal-content">
                <div class="quadtree-modal-header"><h3>Change Cell
Color</h3><button class="quadtree-modal-close">&times;</button></div>
                <div class="quadtree-modal-body">
                    <div class="form-group"><label for="cell-color">Select
Color:</label><input type="color" id="cell-color"
value="${getComputedStyle(cell).backgroundColor || '#f0f0f0'}"></div>
                </div>
                <div class="quadtree-form-actions"><button class="button-
secondary modal-cancel">Cancel</button><button class="modal-apply">Apply</button>
                </div>
            `;
        const modal = showModal('color-picker-modal', modalContent);

        modal.querySelector('.modal-apply').onclick = () => {
            cell.style.backgroundColor = doc.getElementById('cell-
color').value;
            markCellAsCustomized(cell);
            modal.classList.remove('active');
        };
        modal.querySelector('.modal-cancel').onclick = () =>
        modal.classList.remove('active');
        modal.querySelector('.quadtree-modal-close').onclick = () =>
    }
}
```

```
modal.classList.remove('active');
}

function showTextModal(cell) {
    const existingText = cell.querySelector('.quadtree-cell-custom-text')?.textContent || '';
    const modalContent = `
        <div class="quadtree-modal-content">
            <div class="quadtree-modal-header"><h3>Add Text to Cell</h3><button class="quadtree-modal-close">&times;</button></div>
            <div class="quadtree-modal-body">
                <div class="form-group"><label for="cell-text">Text:</label><input type="text" id="cell-text" value="${existingText}"></div>
                <div class="form-group"><label for="text-color">Text Color:</label><input type="color" id="text-color" value="#000000"></div>
            </div>
            <div class="quadtree-form-actions"><button class="button-secondary modal-cancel">Cancel</button><button class="modal-apply">Apply</button>
        </div>
    `;
    const modal = showModal('text-modal', modalContent);

    modal.querySelector('.modal-apply').onclick = () => {
        const text = doc.getElementById('cell-text').value;
        const color = doc.getElementById('text-color').value;
        let textElement = cell.querySelector('.quadtree-cell-custom-text');

        if (text.trim() !== '') {
            if (!textElement) {
                textElement = doc.createElement('div');
                textElement.className = 'quadtree-cell-custom-text';
                cell.appendChild(textElement);
            }
            textElement.textContent = text;
            textElement.style.color = color;
            markCellAsCustomized(cell);
        } else if (textElement) {
            cell.removeChild(textElement);
        }
        modal.classList.remove('active');
    };
    modal.querySelector('.modal-cancel').onclick = () =>
        modal.classList.remove('active');
        modal.querySelector('.quadtree-modal-close').onclick = () =>
            modal.classList.remove('active');
    }

    function handleImageUpload(event) {
        const cell = activeQuadtreeCell;
        if (!cell || !event.target.files || !event.target.files[0])
return;
        const file = event.target.files[0];
        if (!file.type.startsWith('image/')) return;
    }
}
```

```
const reader = new FileReader();
reader.onload = (e) => addImageToCell(cell, e.target.result);
reader.readAsDataURL(file);
event.target.value = '';
}

function addImageToCell(cell, imageDataUrl) {
    let img = cell.querySelector('.quadtree-cell-image');
    if (!img) {
        img = doc.createElement('img');
        img.className = 'quadtree-cell-image';
        cell.appendChild(img);
    }
    img.src = imageDataUrl;
    markCellAsCustomized(cell);
}

function resetCell(cell) {
    const textElement = cell.querySelector('.quadtree-cell-custom-text');
    if (textElement) cell.removeChild(textElement);
    const imageElement = cell.querySelector('.quadtree-cell-image');
    if (imageElement) cell.removeChild(imageElement);
    cell.style.backgroundColor = '';
    cell.classList.remove('customized', 'paste-focus');
}

function prepareCellForPaste(cell) {
    doc.querySelector('.quadtree-cell.paste-focus')?.classList.remove('paste-focus');
    cell.classList.add('paste-focus');
    const indicator = doc.getElementById('paste-indicator');
    indicator.classList.add('active');
    setTimeout(() => indicator.classList.remove('active'), 3000);
}

function setupClipboardPaste() {
    doc.addEventListener('paste', (event) => {
        const cell = doc.querySelector('.quadtree-cell.paste-focus');
        if (!cell) return;
        const items = (event.clipboardData || event.originalEvent.clipboardData).items;
        for (const item of items) {
            if (item.type.indexOf('image') === 0) {
                const blob = item.getAsFile();
                const reader = new FileReader();
                reader.onload = (e) => {
                    addImageToCell(cell, e.target.result);
                    cell.classList.remove('paste-focus');
                    doc.getElementById('paste-indicator').classList.remove('active');
                };
                reader.readAsDataURL(blob);
            }
        }
    });
}
```

```
        event.preventDefault();
        return;
    }
}
});

function exportEnhancedPNG() {
    showLoadingIndicator('Generating PNG...');
    if (typeof html2canvas === 'function') {
        html2canvas(quadtreeContainer, {
            scale: 2,
            backgroundColor: null,
            logging: false,
            useCORS: true,
            allowTaint: true
        }).then(canvas => {
            finishExport(canvas);
        }).catch(error => {
            console.error('html2canvas failed:', error);
            gosApiProxy.ui.showNotification('Error', 'Failed to export
quadtree: ' + error.message, 5000);
            hideLoadingIndicator();
        });
    } else {
        gosApiProxy.ui.showNotification('Error', 'Export library not
found.', 5000);
        hideLoadingIndicator();
    }
}

function finishExport(canvas) {
    try {
        const dataUrl = canvas.toDataURL('image/png');
        const link = doc.createElement('a');
        link.download = 'quadtree_export.png';
        link.href = dataUrl;
        link.click();
    } catch (error) {
        console.error('Error generating PNG:', error);
        gosApiProxy.ui.showNotification('Error', 'Failed to generate
PNG: ' + error.message, 5000);
    } finally {
        hideLoadingIndicator();
    }
}

function showLoadingIndicator(message) {
    let loadingEl = doc.getElementById('export-loading');
    if (!loadingEl) {
        loadingEl = doc.createElement('div');
        loadingEl.id = 'export-loading';
        loadingEl.className = 'export-loading';
        loadingEl.innerHTML =
`
```

```

        <div class="export-loading-content">
            <div class="export-spinner"></div>
            <div>${message}</div>
        </div>`;
        doc.body.appendChild(loadingEl);
    }
    loadingEl.classList.add('active');
}

function hideLoadingIndicator() {
    doc.getElementById('export-loading')?.classList.remove('active');
}

// --- INITIAL SETUP ---
createQuadtreeBtn.addEventListener('click', initQuadtree);
resetQuadtreeBtn.addEventListener('click', initQuadtree);
exportQuadtreeBtn.addEventListener('click', exportEnhancedPNG);

initQuadtree();
initExtensions();

} catch (error) {
    console.error("Failed to initialize Quadtree Visualizer:", error);
    container.innerHTML = `<div style="color: red; padding: 20px;">Error
loading application resources: ${error.message}</div>`;
}
}

// Run the main application setup.
main();
}

```

### [39] GenesisOS/quadtree-visualizer.json

- **Bytes:** 533
- **Type:** text

```
{
    "id": "quadtree-visualizer",
    "title": "Quadtree Visualizer",
    "description": "An interactive tool to create, visualize, and customize
quadtrees. Subdivide cells, add text\\images, and export your creations.",
    "version": "1.0.0",
    "author": "GOS Conversion",
    "icon": "\ud83c\udf33",
    "category": "Development",
    "entry": "quadtree-visualizer.js",
    "permissions": [
        "filesystem.write.user.*"
    ],
    "window": {

```

```
        "width": 1250,
        "height": 700,
        "resizable": true
    }
}
```

## [40] GenesisOS/repl-computer.js

- **Bytes:** 28185
- **Type:** text

```
// repl-computer.js – Accumulator Computer in Genesis OS (v1.4.2)
// Minimal UI + Programmer Panel (Assembler, Memory, Breakpoints)
// ISA: NOP, LDI, ADDI, SUBI, STA, LDA, DSP, HLT, PRT
// Directives: .ORG, .BYTE (numbers, labels, "strings" with escapes)

export function initialize(gosApiProxy){
    const container = gosApiProxy.window.getContainer();
    const doc = container.ownerDocument;

    // ----- Styles -----
    const style = doc.createElement('style');
    style.textContent = [
        ":root {",
        "  --bg-color: #1e1e2e;",
        "  --editor-bg: #282a36;",
        "  --text-color: #cdd6f4;",
        "  --muted: #6c7086;",
        "  --border-color: #313244;",
        "  --focus-color: #f5c2e7;",
        "  --scrollbar-bg: #313244;",
        "  --scrollbar-thumb: #585b70;",
        "  --err: #e74c3c;",
        "  --ok: #70d392;",
        "  --topbar-h: 36px;",
        "}",
        ".gos-root{font-family:'Fira Code','Cascadia Code','JetBrains Mono',monospace;",
        "  background:var(--bg-color); min-height:100%; display:flex; align-items:center; justify-content:center;",
        "  padding:20px; position:relative; color:var(--text-color)}",
        ".editor{width:94%; max-width:940px; height:76vh; background:var(--editor-bg);",
        "  border:2px solid var(--border-color); border-radius:8px; box-shadow:0 4px 30px rgba(0,0,0,.3);",
        "  display:grid; grid-template-rows: var(--topbar-h) 1fr; overflow:hidden;",
        "  transition:border-color .3s ease, box-shadow .3s ease; position:relative}",
        ".editor:focus-within{outline:none; border-color:var(--focus-color);",
        "  box-shadow:0 0 0 3px rgba(245,194,231,.3), 0 4px 30px rgba(0,0,0,.3);}",
        ".topbar{grid-row:1/2; display:flex; align-items:center; gap:8px; padding:6px 10px;}",
    ]
}
```

```
" border-bottom:1px solid var(--border-color);",
" background:linear-gradient(180deg,rgba(255,255,255,.02),rgba(0,0,0,0)); z-
index:3}",
".shell{grid-row:2/3; display:flex; overflow:hidden}",
".left{min-width:45px; border-right:1px solid var(--border-color); color:var(-
-muted);",
" padding:15px 10px 15px 15px; line-height:1.6; user-select:none;
overflow:auto; white-space:pre}",
".right{flex:1; display:flex; flex-direction:column; min-width:0}",
".log{flex:1; padding:12px 15px; overflow:auto; line-height:1.6; white-
space:pre-wrap}",
".input{border-top:1px solid var(--border-color); display:flex; gap:8px;
padding:10px}",
".cmd{flex:1; background:transparent; color:var(--text-color); border:none;
outline:none; font-size:16px}",
".cmd::placeholder{color:var(--muted)}",
".tag{display:inline-block; border:1px solid var(--border-color); padding:2px
6px; border-radius:999px; color:#aeb3cf; margin-right:6px}",
".ok{color:var(--ok)} .err{color:var(--err)} .muted{color:var(--muted)}",
".pillbtn{display:inline-flex; align-items:center; gap:6px; padding:4px 8px;
border:1px solid var(--border-color); border-radius:999px; cursor:pointer;
background:#1f2023; color:#bfc3df}",
".pillbtn:hover{filter:brightness(1.05)}",
".panel{position:absolute; right:0; top:var(--topbar-h); height:calc(100% -
var(--topbar-h)); width:0; overflow:hidden;",
" transition:width .25s ease; border-left:1px solid var(--border-color);
background:var(--editor-bg); z-index:2}",
".panel.open{width:min(54%,520px)}",
".panel-inner{display:flex; flex-direction:column; height:100%}",
".tabs{display:flex; gap:6px; border-bottom:1px solid var(--border-color);
padding:8px}",
".tab{padding:6px 10px; border:1px solid var(--border-color); border-
bottom:none; border-top-left-radius:8px; border-top-right-radius:8px;
background:#222432; color:#cf3f4; cursor:pointer}",
".tab.active{background:#1e2030}",
".pane{flex:1; display:none; min-height:0}",
".pane.active{display:flex; overflow:auto}",
".asm-left{flex:1; display:flex; flex-direction:column; border-right:1px solid
var(--border-color); min-height:0}",
".asm-src{flex:1; background:transparent; color:var(--text-color);
border:none; outline:none; padding:10px; resize:none; font-size:14px; line-
height:1.5; min-height:0; overflow:auto}",
".asm-actions{display:flex; gap:8px; padding:8px; border-top:1px solid var(---
border-color)}",
".asm-right{width:180px; padding:8px; display:flex; flex-direction:column;
gap:8px}",
".mini-input{width:100%; background:#202233; color:#e9ecff; border:1px solid
var(--border-color); border-radius:8px; padding:6px 8px; outline:none}",
".btn{background:#202124; color:#f1f1f1; border:1px solid var(--border-color);
padding:7px 10px; border-radius:8px; cursor:pointer}",
".btn:hover{filter:brightness(1.06)}",
".mem-grid{flex:1; overflow:auto; padding:8px}",
"table.mem{border-collapse:collapse; font-size:12px; width:100%}",
"table.mem th, table.mem td{border:1px solid var(--border-color); padding:4px};
```



```
'          <label>PC after assemble</label>',
'          <input id="asmPC" class="mini-input" value="0x00"/>',
'          <div class="muted" style="font-size:12px">Supported: NOP, LDI
#imm, ADDI #imm, SUBI #imm, STA addr, LDA addr, DSP, HLT, .BYTE list
(numbers/labels/strings). Labels allowed.</div>',
'          </div>',
'          </div>',
'          <div class="pane" id="pane-mem">',
'            <div class="mem-grid" id="memGrid"></div>',
'          </div>',
'          <div class="pane" id="pane-bp">',
'            <div class="flex" style="padding:8px">,
'              <input id="bpAddr" class="mini-input" placeholder="0x00"/>,
'              <button class="btn" id="bpAdd">Add</button>,
'              <button class="btn" id="bpClear">Clear All</button>,
'            </div>,
'            <div class="bp-list" id="bpList"></div>,
'          </div>',
'        </div>,
'      </div>,
'    </div>,
'    <div class="hint">Type <code>:help</code> for documentation</div>'  
].join("\n");
container.appendChild(root);

// -----
const $ = s => root.querySelector(s);
const logEl = $("#log"), linesEl = $("#lines"), input = $("#cmd"), panel =
$("#panel");
$("#panelBtn").addEventListener("click", ()=> panel.classList.toggle("open"));
root.addEventListener("click", (e)=>{
  if(e.target.classList.contains("tab")){
    root.querySelectorAll(".tab").forEach(t=>t.classList.remove("active"));
    e.target.classList.add("active");
    root.querySelectorAll(".pane").forEach(p=>p.classList.remove("active"));
    $("#pane-"+e.target.dataset.pane).classList.add("active");
  }
});
// Optional keyboard toggle (Ctrl/Cmd+P)
root.addEventListener("keydown", (e)=>{
  if((e.ctrlKey|e.metaKey) && e.key.toLowerCase() === "p"){ e.preventDefault();
  panel.classList.toggle("open"); }
});

let lineCount = 1;
const print = (msg, cls="") => {
  const div = doc.createElement("div");
  if(cls) div.className = cls;
  div.textContent = msg;
  logEl.appendChild(div);
  logEl.scrollTop = logEl.scrollHeight;
  lineCount += String(msg).split("\n").length;
  linesEl.textContent = Array.from({length:lineCount}, (_,i)=>i+1).join("\n");
```

```
};

const setTag = (id, text) => { const el = `#${id}`; if(el) el.textContent = text; };

// ----- CPU/BUS Model -----
const clamp8 = x => ((x % 256) + 256) % 256;
const toHex2 = x => x.toString(16).toUpperCase().padStart(2, "0");
function Latch(name){ this.name=name; this.q=0; this.next=0; this.enable=true;
  this.latch=(d)=>{ if(this.enable){ this.next = clamp8(d); } };
  this.clock=()=>{ this.q = this.next; };
  function add8_ripple(a,b){ let c=0,o=0; for(let i=0;i<8;i++){ const A=
  (a>>i)&1,B=(b>>i)&1; const s=A^B^c; const m=(A&B)|(A&c)|(B&c); o|=(s<<i); c=m&1; }
  return o&0xFF; }
  function sub8_ripple(a,b){ return add8_ripple(a, add8_ripple(~b)&0xFF, 1)); }

const ACC = new Latch("ACC");
const IR = new Latch("IR");
const DR = new Latch("DR");
const PC = new Latch("PC");
const CTRL = { developer:false };

const MEM = new Uint8Array(256);
const loadMem = (bytes, at=0)=>{ for(let i=0;i<bytes.length;i++)
MEM[(at+i)&0xFF]=bytes[i]&0xFF; renderMem(); };

const PHASES = ["T0","T1","T2","T3","T4"];
let phase = 0;
let auto = false;
let hz = 2;
let timer = null;
const breakpoints = new Set();

// PRT microcode state
let prtActive = false;
let prtPtr = 0;

const updateTop = () => {
  setTag("modeTag", "MODE: " + (auto? "auto":"manual"));
  setTag("clockTag", "CLK: " + (timer? "running":"stopped"));
  setTag("hzTag", "HZ: " + hz);
  setTag("phaseTag", "PHASE: " + PHASES[phase]);
  setTag("pcTag", "PC: " + toHex2(PC.q));
  setTag("accTag", "ACC: " + toHex2(ACC.q));
};

function decoder4AND(op, en){
  const lines = new Array(16).fill(0);
  if(!en) return lines;
  const b0=(op>>0)&1, b1=(op>>1)&1, b2=(op>>2)&1, b3=(op>>3)&1;
  for(let i=0;i<16;i++){
    const i0=(i>>0)&1,i1=(i>>1)&1,i2=(i>>2)&1,i3=(i>>3)&1;
    lines[i]=(b0==i0 && b1==i1 && b2==i2 && b3==i3)?1:0;
  }
  return lines;
}
```

```

}

const OPC = { NOP:0x0, LDI:0x1, ADDI:0x2, SUBI:0x3, STA:0x4, LDA:0x5, DSP:0x6,
HLT:0x7, PRT:0x8 };

function stepMicro(){
// PRT loop
if(prtActive){
    switch(phase){
        case 0: DR.latch(MEM[prtPtr]); DR.clock(); break;
        case 1:
            if(DR.q==0){ prtActive=false; print("PRT done","muted"); }
            else { const ch = String.fromCharCode(DR.q); print("PRT: '" + ch + "'"
(0x" + DR.q.toString(16).toUpperCase().padStart(2,"0") + ")"); }
            break;
        case 2: if(DR.q!=0){ prtPtr = (prtPtr + 1) & 0xFF; } break;
        case 3: case 4: break;
    }
    phase = (phase+1)%5; updateTop(); return;
}

const op = IR.q & 0x0F;
const onehot = decoder4AND(op, true);
switch(phase){
    case 0: // fetch
        DR.latch(MEM[PC.q]); DR.clock();
        PC.latch(PC.q+1); PC.clock();
        IR.latch(DR.q); IR.clock();
        break;
    case 1: // operand fetch / PRT init

if(onehot[OPC.LDI] || onehot[OPC.ADDI] || onehot[OPC.SUBI] || onehot[OPC.LDA] || onehot[OPC.STA]){
    DR.latch(MEM[PC.q]); DR.clock(); PC.latch(PC.q+1); PC.clock();
}
if(onehot[OPC.PRT]){
    prtActive = true; prtPtr = ACC.q;
    break;
}
case 2:
    if(onehot[OPC.LDI]){ ACC.latch(DR.q); ACC.clock(); }
    if(onehot[OPC.LDA]){ ACC.latch(MEM[DR.q]); ACC.clock(); }
    break;
case 3:
    if(onehot[OPC.ADDI]){ ACC.latch(add8_ripple(ACC.q, DR.q)); ACC.clock(); }
    if(onehot[OPC.SUBI]){ ACC.latch(sub8_ripple(ACC.q, DR.q)); ACC.clock(); }
    if(onehot[OPC.STA]){ MEM[DR.q] = ACC.q; renderMemCell(DR.q); }
    break;
case 4:
    if(onehot[OPC.DSP]){ print("DISPLAY: " + toHex2(ACC.q) + " (" + ACC.q +
")", "ok"); }
    if(onehot[OPC.HLT]){ stopClock(); print("HALT","muted"); }
    break;
}
phase = (phase+1)%5;
updateTop();
}

```

```
if(phase==0 && breakpoints.has(PC.q)){ stopClock(); print("BREAK @ " +  
toHex2(PC.q), "err"); }  
  
function tick(){ stepMicro(); }  
function startClock(){ if(timer) return; timer = setInterval(tick, Math.max(5,  
1000/Math.max(1,hz))); updateTop(); }  
function stopClock(){ if(timer){ clearInterval(timer); timer=null; }  
updateTop(); }  
function setHz(v){ hz = Math.max(1, Math.min(1000, v|0)); if(timer){  
stopClock(); startClock(); } updateTop(); }  
  
// ----- Programmer Panel: Memory & Breakpoints -----  
function renderMem(){  
const host = $("#memGrid"); if(!host) return;  
const table = doc.createElement("table"); table.className="mem";  
const thead = doc.createElement("thead");  
const hr = doc.createElement("tr");  
hr.appendChild(thead);  
for(let x=0;x<16;x++) hr.appendChild(th(x.toString(16).toUpperCase()));  
thead.appendChild(hr); table.appendChild(thead);  
const tbody = doc.createElement("tbody");  
for(let r=0;r<16;r++){  
const tr = doc.createElement("tr");  
tr.appendChild(th((r*16).toString(16).toUpperCase().padStart(2,"0")));  
for(let c=0;c<16;c++){  
const addr = r*16+c;  
const td = doc.createElement("td");  
const inp = doc.createElement("input"); inp.className="cell"; inp.value =  
toHex2(MEM[addr]);  
inp.maxLength = 2;  
inp.addEventListener("change", ()=>{  
const v = parseInt(inp.value,16);  
if(isNaN(v)){ inp.value = toHex2(MEM[addr]); return; }  
MEM[addr]=v&0xFF; inp.value = toHex2(MEM[addr]);  
});  
td.appendChild(inp); tr.appendChild(td);  
}  
tbody.appendChild(tr);  
}  
table.appendChild(tbody);  
host.innerHTML = ""; host.appendChild(table);  
}  
function renderMemCell(addr){  
const host = $("#memGrid"); if(!host) return;  
const inp = host.querySelectorAll("input.cell")[addr];  
if(inp) inp.value = toHex2(MEM[addr]);  
}  
function th(t){ const e=doc.createElement("th"); e.textContent=t; return e; }  
  
function renderBreakpoints(){  
const list = $("#bpList"); list.innerHTML = "";  
if(breakpoints.size==0){ const d=doc.createElement("div");  
d.className="muted"; d.textContent="(no breakpoints)"; list.appendChild(d);  
}
```

```
return; }

[...breakpoints].sort((a,b)=>a-b).forEach(addr=>{
    const row = doc.createElement("div"); row.className="flex";
    const tag = doc.createElement("span"); tag.className="tag";
    tag.textContent="@"+toHex2(addr);
    const del = doc.createElement("button"); del.className="btn";
    del.textContent="Remove";
    del.onclick = ()=>{ breakpoints.delete(addr); renderBreakpoints(); };
    row.append(tag, del); list.appendChild(row);
});

}

$("#bpAdd").addEventListener("click", ()=>{
    const v = $("#bpAddr").value.trim();
    const val = /^0x/i.test(v) ? parseInt(v,16) : parseInt(v,16);
    if(isNaN(val)){ print("invalid breakpoint addr","err"); return; }
    breakpoints.add(val & 0xFF); renderBreakpoints();
});
$("#bpClear").addEventListener("click", ()=>{ breakpoints.clear();
renderBreakpoints(); });

// ----- Assembler -----
const ASM = {
    assemble(src, origin){
        // Two-pass with labels + .BYTE (numbers, labels, strings with escapes)
        const lines = src.split(/\r?\n/);
        const rows = lines.map((ln,i)=>({i, raw:ln, t:
ln.replace(/;.*$/,"").trim()})).filter(x=>x.t.length>0);
        const labels = new Map();
        let pc = origin|0;

        const isOrg = (t) => /^\.org\b/i.test(t);
        const isByte = (t) => /^\.byte\b/i.test(t);

        const parseByteList = (arg, labels) => {
            const tokens = [];
            let i=0, inQ=false, qch='', esc=false, cur="";
            while(i<arg.length){
                const ch = arg[i++];
                if(inQ){
                    if(esc){ cur += ch; esc=false; continue; }
                    if(ch==='\\'){ esc=true; continue; }
                    if(ch==qch){ tokens.push({type:"str", s:cur}); cur=""; inQ=false;
continue; }
                    cur += ch;
                } else {
                    if(ch===''' || ch=='''') { inQ=true; qch=ch; continue; }
                    if(ch==',' || /\s/.test(ch)){
                        if(cur.trim().length){ tokens.push({type:"num", s:cur.trim()});
cur=""; }
                        continue;
                    }
                    cur += ch;
                }
            }
        }
    }
}
```

```

    }

    if(cur.trim().length){ tokens.push({type:"num", s:cur.trim()}); }

    const unescape = (s) => {
        const out = [];
        for(let i=0;i<s.length;i++){
            let ch = s[i];
            if(ch==='\\' && i+1<s.length){
                const n = s[++i];
                if(n==='n'){ out.push(0x0A); continue; }
                if(n==='r'){ out.push(0x0D); continue; }
                if(n==='t'){ out.push(0x09); continue; }
                if(n==='0'){ out.push(0x00); continue; }
                if(n==='\\'){ out.push(0x5C); continue; }
                if(n==='"'){ out.push(0x22); continue; }
                if(n==='\''){ out.push(0x27); continue; }
                if(n==='x' && i+2<s.length){
                    const hh = s.substring(i+1, i+3);
                    const v = parseInt(hh,16);
                    if(!isNaN(v)){ out.push(v&0xFF); i+=2; continue; }
                }
                out.push(n.charCodeAt(0));
                continue;
            }
            out.push(ch.charCodeAt(0));
        }
        return out;
    };

    const toByte = (s)=>{
        if(s==null || s==='') return 0;
        if(/^#/i.test(s)) s = s.replace(/^#/,"");
        if(/^0x/i.test(s)) return (parseInt(s,16)&0xFF);
        if(/^[-0-9]+$/i.test(s)) return (parseInt(s,10)&0xFF);
        if(labels.has(s)) return (labels.get(s)&0xFF);
        throw new Error("Unknown number/label: "+s);
    };

    const out = [];
    tokens.forEach(t=>{
        if(t.type==="str"){ unescape(t.s).forEach(b=> out.push(b&0xFF)); }
        else { out.push(toByte(t.s)); }
    });
    return out;
};

// Pass 1
rows.forEach(x=>{
    const m = x.t.match(/^([A-Za-z_]\w*):\s*(.*)$/);
    if(m){ labels.set(m[1], pc & 0xFF); x.t = m[2].trim(); }
    if(!x.t) return;

    if(isOrg(x.t)){
        const arg = x.t.replace(/^\.\org\b/i,"").trim();

```

```

pc = parseInt(arg.replace(/^0x/i,""), 16) & 0xFF;
return;
}
if(isByte(x.t)){
  const arg = x.t.replace(/^\.\byte\b/i,"").trim();
  try { const bytes = parseByteList(arg, labels); pc = (pc + bytes.length)
& 0xFF; } catch(e){}
  return;
}

const opcode = x.t.split(/\s+/)[0].toUpperCase();
if(["LDI","ADDI","SUBI","STA","LDA"].includes(opcode)) pc = (pc + 2) &
0xFF;
else if(["NOP","DSP","HLT","PRT"].includes(opcode)) pc = (pc + 1) & 0xFF;
else if(opcode) throw new Error("Unknown opcode/directive on pass1:
"+opcode);
});

// Pass 2
pc = origin|0;
const out = [];
const toByte = (s)=>{
  if(s==null || s==="") return 0;
  if(/^\#/test(s)) s = s.replace(/^\#/,"");
  if(/^\0x/i.test(s)) return (parseInt(s,16)&0xFF);
  if(/^\[0-9]+\$/test(s)) return (parseInt(s,10)&0xFF);
  if(labels.has(s)) return (labels.get(s)&0xFF);
  throw new Error("Unknown number/label: "+s);
};

rows.forEach(x=>{
  if(!x.t) return;

  if(isOrg(x.t)){
    const arg = x.t.replace(/^\.\org\b/i,"").trim();
    pc = parseInt(arg.replace(/^\0x/i,""), 16) & 0xFF;
    return;
  }
  if(isByte(x.t)){
    const arg = x.t.replace(/^\.\byte\b/i,"").trim();
    const bytes = parseByteList(arg, labels);
    bytes.forEach(b=> out.push(b&0xFF));
    pc = (pc + bytes.length) & 0xFF;
    return;
  }

  const parts = x.t.split(/\s+/);
  const op = parts[0].toUpperCase();
  const arg = parts.slice(1).join(" ").trim();

  switch(op){
    case "NOP": out.push(0x00); pc=(pc+1)&0xFF; break;
    case "LDI": out.push(0x01, toByte(arg)); pc=(pc+2)&0xFF; break;
    case "ADDI": out.push(0x02, toByte(arg)); pc=(pc+2)&0xFF; break;
  }
}

```

```

        case "SUBI": out.push(0x03, toByte(arg)); pc=(pc+2)&0xFF; break;
        case "STA": out.push(0x04, toByte(arg)); pc=(pc+2)&0xFF; break;
        case "LDA": out.push(0x05, toByte(arg)); pc=(pc+2)&0xFF; break;
        case "DSP": out.push(0x06); pc=(pc+1)&0xFF; break;
        case "HLT": out.push(0x07); pc=(pc+1)&0xFF; break;
        case "PRT": out.push(0x08); pc=(pc+1)&0xFF; break;
        default: throw new Error("Unknown opcode on pass2: "+op);
    }
});
return out;
}

// ----- REPL -----
const help = [
    "OVERVIEW",
    " Accumulator CPU with 5-phase microcycle (T0..T4), 4-AND decoder (4-bit), 8-pin bus model, latches, ripple adder.",
    " Manual clock (:tick) and automatic clock (:clock start / :clock stop / :hz N).",
    " Admin control (:admin on|off) gates privileged ops.",
    " Programmer Panel: :panel to toggle – Assembler, Memory Inspector, Breakpoints.",
    "",
    "INSTRUCTIONS",
    " NOP(00) LDI(01) imm8 ADDI(02) imm8 SUBI(03) imm8 STA(04) addr8 LDA(05) addr8 DSP(06) HLT(07) PRT(08)",
    " PRT: print zero-terminated string starting at ACC via a microcoded loop",
    "",
    "DIRECTIVES",
    " .BYTE <items...> # hex (0x..), decimal, label, or \\\"string\\\" with escapes; emits bytes at current PC",
    " .ORG <addr> # set assembly origin",
    "",
    "REPL COMMANDS",
    " :help",
    " :panel           Toggle programmer panel",
    " :admin on|off    Toggle developer control unit",
    " :mode manual|auto Set manual or automatic clock mode",
    " :clock start|stop Start/stop the clock (auto mode)",
    " :hz <n>          Set auto clock frequency (1..1000 Hz)",
    " :tick             Single clock tick",
    " :stepi            Run to next instruction boundary (phase T0)",
    " :state             Dump CPU state",
    " :mem set <addr> <val> Set MEM[addr]=val (hex)",
    " :load <hex bytes> Load program at 0x00 (space/comma separated)",
    " :pc <hex>          Set program counter",
    " :run <n>           Tick n cycles (manual mode)",
    " :acc set <n>       Directly set ACC (admin only, decimal)",
    " :break add <addr>  Add breakpoint (hex)",
    " :break del <addr> Remove breakpoint",
    " :break list         List breakpoints",
    " :reset              Reset registers and phase (memory preserved)"
].join("\n");

```

```

    function parseHexBytes(s){ return s.replace(/[,]/g,"")
").trim().split(/\s+/).filter(Boolean).map(x=>parseInt(x,16)&0xFF); }
    function dumpState(){ print("PC="+toHex2(PC.q)+" ACC="+toHex2(ACC.q)+""
IR="+toHex2(IR.q)+" DR="+toHex2(DR.q)+" PHASE="+PHASES[phase]); }
    function requireAdmin(){ if(!CTRL.developer){ print("admin-developer required.
Use :admin on","err"); return false; } return true; }

    function stepi(){ const startPhase = phase; do { stepMicro(); } while(phase !==
0); print("stepi","muted"); }

    function runCmd(line){
        const raw = line.trim(); if(!raw) return;
        if(raw[0] !== ":"){ print("(data) "+raw,"muted"); return; }
        const [cmd, ...rest] = raw.split(/\s+/);
        const args = rest.join(" ");

        switch(cmd){
            case ":help": print(help); break;
            case ":panel": panel.classList.toggle("open"); break;
            case ":admin": CTRL.developer = /^on$/i.test(rest[0])||"";;
print("developer="+(CTRL.developer?"on":"off"),"ok"); break;
            case ":mode": if(/^auto$/i.test(rest[0])||""){ auto=true; startClock(); }
else { auto=false; stopClock(); } updateTop(); print("mode="+
(auto?"auto":"manual"),"ok"); break;
            case ":clock": if(/^start$/i.test(rest[0])||""){ auto=true; startClock(); }
print("clock started","ok"); } else { stopClock(); auto=false; print("clock
stopped","muted"); } break;
            case ":hz": setHz(parseInt(rest[0]||"2",10)||2); print("hz="+hz,"ok");
break;
            case ":tick": if(auto){ print("stop auto clock first (:clock stop)","err");
break; } tick(); print("tick","muted"); break;
            case ":stepi": if(auto){ print("stop auto clock first (:clock stop)","err");
break; } stepi(); break;
            case ":state": dumpState(); break;
            case ":mem":
                if(rest[0]==="set"){ const addr=parseInt(rest[1],16)&0xFF; const
val=parseInt(rest[2],16)&0xFF; MEM[addr]=val; renderMemCell(addr);
print("MEM["+toHex2(addr)+"]="+toHex2(val),"ok"); }
                else print("usage: :mem set <addrHEX> <valHEX>","muted"); break;
            case ":load": {
                const bytes = parseHexBytes(args);
                for(let i=0;i<256;i++) MEM[i]=0;
                for(let i=0;i<bytes.length;i++) MEM[i]=bytes[i];
                renderMem();
                PC.q=0; IR.q=0; DR.q=0; ACC.q=0; phase=0;
                print("program loaded ("+bytes.length+" bytes) at 00","ok"); dumpState();
            } break;
            case ":pc": PC.q = parseInt(rest[0]||"0",16)&0xFF; updateTop(); dumpState();
break;
            case ":run": {
                const n = Math.max(0, parseInt(rest[0]||"0",10)|0); if(auto){ print("stop
auto clock first (:clock stop)","err"); break; }
                for(let i=0;i<n;i++) tick(); print("ran "+n+" ticks","muted");
            }
        }
    }
}

```

```

        } break;
    case ":acc":
        if(rest[0]==="set"){ if(!requireAdmin()) break; ACC.q =
clamp8(parseInt(rest[1]||"0",10)|0); updateTop();
print("ACC="+toHex2(ACC.q),"ok"); }
        else print("usage: :acc set <decimal> (admin)","muted"); break;
    case ":break":
        if(rest[0]==="add"){ const a=parseInt(rest[1],16)&0xFF;
breakpoints.add(a); renderBreakpoints(); print("break @ "+toHex2(a),"ok"); }
        else if(rest[0]==="del"){ const a=parseInt(rest[1],16)&0xFF;
breakpoints.delete(a); renderBreakpoints(); print("break removed @
"+toHex2(a),"ok"); }
        else if(rest[0]==="list"){ print("breakpoints: "+
([...breakpoints].map(x=>toHex2(x)).join(", ")||"(none)")); }
        else print("usage: :break add|del|list <addrHEX?>","muted");
        break;
    case ":reset":
        PC.q=0; IR.q=0; DR.q=0; ACC.q=0; phase=0; updateTop();
print("reset","muted"); break;
    default: print("unknown command: "+cmd+". Type :help","err");
}
}

// -----
(function boot(){
    const demo = [0x01, 0x2A, 0x06, 0x07]; // LDI 2A; DSP; HLT
    for(let i=0;i<demo.length;i++) MEM[i]=demo[i];
    PC.q=0; ACC.q=0; IR.q=0; DR.q=0; phase=0;
})();
print("REPL Computer ready. Type :help");
print("Tip: PRT (0x08) prints a zero-terminated string from ACC");
dumpState(); updateTop(); renderMem(); renderBreakpoints();

// -----
input.addEventListener("keydown", (e)=>{
    if(e.key==="Enter"){
        e.preventDefault();
        const v = input.value; input.value="";
        print("> "+v,"muted"); runCmd(v);
    }
});
}
}

```

## [41] GenesisOS/repl-computer.json

- **Bytes:** 464
- **Type:** text

```
{
    "id": "repl-computer",
```

```

"title": "REPL Computer (Accumulator CPU)",
"description": "Accumulator-based REPL computer with programmer panel, .BYTE strings, and PRT (zero-terminated string print).",
"version": "1.4.1",
"author": "You",
"icon": "\ud83e\uddee",
"category": "Development",
"entry": "repl-computer.js",
"permissions": [],
>window": {
    "width": 1000,
    "height": 720,
    "resizable": true
}
}

```

#### [42] GenesisOS/sandbox.html

- **Bytes:** 232
- **Type:** text

```

<!DOCTYPE html>
<html>
<head>
    <title>GOS Sandbox</title>
    <style>
        /* Ensures the sandboxed app can fill the frame */
        body { margin: 0; padding: 0; overflow: hidden; }
    </style>
</head>
<body>
    </body>
</html>

```

#### [43] GenesisOS/sandbox.php

- **Bytes:** 5751
- **Type:** text

```

<?php
/**
 * Text Reversion
 * Sandbox Module
 */
/* =====
 * SANDBOX MODULE
 * ===== */

```

```
* Handle sandbox API requests
*/
function handleSandboxAPI() {
    global $config;

    // Use the JSON shim that's already processed the request body
    $method = $_POST['method'] ?? null;
    $params = $_POST['params'] ?? [];

    // Parse params if it's a JSON string
    if (is_string($params)) {
        $params = json_decode($params, true) ?? [];
    }

    // Only admins or users with specific permissions can execute sandboxed code
    $user = getCurrentUser();
    $allowedSandbox = isAdmin() || (isset($params['context']) &&
$params['context'] === 'user');

    if (!$allowedSandbox) {
        return ['success' => false, 'message' => 'Access denied for sandbox
execution'];
    }

    switch ($method) {
        case 'execute':
            if (isset($params['code']) && is_string($params['code'])) {
                $code = $params['code'];
                $context = $params['context'] ?? 'default';

                // Set strict limits for sandboxed code
                $memoryLimit = $config['security']['sandbox_memory_limit'];
                $timeLimit = $config['security']['sandbox_time_limit'];

                // Create a temporary file to execute
                $tempFile = tempnam($config['filesystem']['temp_dir'],
'sandbox_');

                // Sanitize the code - make sure it can't break out of the sandbox
                $code = str_replace('>', '', $code); // Remove PHP closing tag

                // Use a safer approach - create a separate file for the user code
                $userCodeFile = tempnam($config['filesystem']['temp_dir'],
'user_code_');

                file_put_contents($userCodeFile, $code);

                // Create a wrapper that catches errors and limits execution
                $wrapper = <<<'EOT'
<?php
// Set resource limits
ini_set('memory_limit', "{$memoryLimit}");
set_time_limit({$timeLimit});

// Capture output
```

```
ob_start();

// Define safe context variables if needed
$context = "{$context}";
$user = "{$USERNAME}";

// Run the user code in a try-catch block
try {
    // Include the user code file
    include "{$UserCodeFile}";
} catch (Throwable $e) {
    echo "Error: " . $e->getMessage();
}

// Get and clean output
$output = ob_get_clean();
echo json_encode(['output' => $output]);
EOT;

        // Make the wrapper template safer by replacing variables
        $wrapper = str_replace(
            ['{$memoryLimit}', '{$timeLimit}', "{$context}",
            "{$USERNAME}", "{$UserCodeFile}"],
            [$memoryLimit, $timeLimit, $context, ($user['username'] ???
            'anonymous'), $UserCodeFile],
            $wrapper
        );

        // Write the wrapper to the temp file
        file_put_contents($tempFile, $wrapper);

        // Execute in a separate process
        $descriptorspec = [
            0 => ["pipe", "r"], // stdin
            1 => ["pipe", "w"], // stdout
            2 => ["pipe", "w"] // stderr
        ];

        // Use disable_functions in the command line for extra safety
        $disableFunctions =
'system,exec,shell_exec,passthru,proc_open,popen,curl_exec,fsockopen';
        $process = proc_open("php -d disable_functions=$disableFunctions
$tempFile", $descriptorspec, $pipes);

        if (is_resource($process)) {
            // Close stdin
            fclose($pipes[0]);

            // Get output
            $output = stream_get_contents($pipes[1]);
            $errors = stream_get_contents($pipes[2]);

            // Close pipes
            fclose($pipes[1]);
```

```

fclose($pipes[2]);

// Close process
$exitCode = proc_close($process);

// Clean up temp files
@unlink($tempFile);
@unlink($userCodeFile);

// Parse the output (should be JSON)
$result = json_decode($output, true);

if ($result !== null) {
    return [
        'success' => true,
        'data' => [
            'output' => $result['output'],
            'errors' => $errors,
            'exitCode' => $exitCode
        ]
    ];
}

return [
    'success' => false,
    'message' => 'Failed to parse sandbox output',
    'data' => [
        'raw' => $output,
        'errors' => $errors,
        'exitCode' => $exitCode
    ]
];
}

return ['success' => false, 'message' => 'Failed to create sandbox
process'];
}
break;
}

return ['success' => false, 'message' => 'Invalid method or parameters'];
}

```

#### [44] GenesisOS/security\_bootstrap.php

- **Bytes:** 3377
- **Type:** text

```

<?php
/**
 * Text Reversion

```

```
* Security Bootstrap Module
*
* Handles HTTPS enforcement, security headers, and secure session management.
* This should be included at the very beginning of the main entry point
(index.php).
*/

// --- 1. HTTPS Enforcement ---
// If the connection is not over HTTPS, redirect to the HTTPS version of the URL.
// This check is skipped for command-line interface (CLI) execution.
if (empty($_SERVER['HTTPS']) || $_SERVER['HTTPS'] === 'off') {
    if (php_sapi_name() !== 'cli') {
        $location = 'https://'.($_SERVER['HTTP_HOST'] ?? 'localhost') .
($_SERVER['REQUEST_URI'] ?? '/');
        header('HTTP/1.1 301 Moved Permanently');
        header('Location: ' . $location);
        exit;
    }
}

// --- 2. Security Headers ---
// These headers are sent to the browser to enable security features.
// They must be sent before any other output.
if (!headers_sent()) {
    // Prevents the site from being rendered in an <iframe>, protecting against
    clickjacking.
    header('X-Frame-Options: SAMEORIGIN');
    // Prevents the browser from MIME-sniffing the content type.
    header('X-Content-Type-Options: nosniff');
    // Enables the XSS protection filter in older browsers.
    header('X-XSS-Protection: 1; mode=block');
    // Sets a basic Content Security Policy. This is configured to allow the
    existing
    // inline scripts and styles to function correctly.
    header("Content-Security-Policy: default-src 'self'; script-src 'self'
unsafe-inline'; style-src 'self' 'unsafe-inline';");
    // HTTP Strict Transport Security (HSTS): Tells the browser to always use
    HTTPS.
    header('Strict-Transport-Security: max-age=31536000; includeSubDomains');
}

// --- 3. Secure Session Configuration ---
// Configures session cookies with security best practices.
session_set_cookie_params([
    'lifetime' => 0, // Session cookie expires when the browser is closed.
    'path' => '/',
    'domain' => $_SERVER['HTTP_HOST'] ?? 'localhost',
    'secure' => true, // The cookie will only be sent over HTTPS.
    'httponly' => true, // The cookie cannot be accessed by JavaScript.
    // --- MODIFICATION START ---
    // Reverted SameSite policy to 'Lax' from 'Strict'. 'Lax' is a strong security
    // default that is less likely to cause issues with session handling than
    'Strict'.
    'samesite' => 'Lax'
```

```

// --- MODIFICATION END ---
]);

// --- 4. Session Start and Validation ---
session_start();

// --- MODIFICATION START ---
// The user fingerprinting check has been commented out. While it adds a layer of
// security, it can be overly sensitive to changes in a user's IP address or
// browser user-agent string, causing the session to be invalidated unexpectedly.
// This was the likely cause of the "failed to load applications" error.
/*
if (isset($_SESSION['user_fingerprint'])) {
    $current_fingerprint = md5($_SERVER['REMOTE_ADDR'] ?? '' .
    $_SERVER['HTTP_USER_AGENT'] ?? ''));

    if ($_SESSION['user_fingerprint'] !== $current_fingerprint) {
        // The fingerprint has changed, which could indicate a session hijack
        attempt.

        // We destroy the session for security.
        session_unset();
        session_destroy();
        // And start a fresh, clean session.
        session_start();
    }
}
*/
// --- MODIFICATION END ---

```

## [45] GenesisOS/solution.cpp

- **Bytes:** 14188
- **Type:** text

```

/**
 * @file solution.cpp
 * @brief Permutation Generator in C++, architected to safety-critical standards.
 *
 * @author Dominic Alexander Cooper (Original Algorithm)
 * @author Gemini / ChatGPT (C++ Architectural Refactoring)
 *
 * @details
 * This program generates all possible character combinations for a given set
 * of lengths, based on a character set provided at runtime.
 *
 * It is designed to align with principles of safety-critical software design,
 * including robust error checking, bounds checking, and abstraction of
 * I/O operations using a C++ interface.
 *
 * This version creates a separate .txt file for each permutation length
 * requested. For example, running:

```

```
*  
*     solution.exe "@dom98" 1 2  
*  
* will generate:  
*  
*     permutations_len_1.txt  
*     permutations_len_2.txt  
*  
* containing all permutations of length 1 and 2 respectively over the chosen  
* alphabet.  
*/  
  
#include <iostream> // For std::cout, std::cerr, std::endl  
#include <fstream> // For std::ofstream  
#include <string> // For std::string  
#include <sstream> // For std::stringstream (filename generation)  
#include <cstdint> // For std::uint64_t  
#include <limits> // For std::numeric_limits  
#include <stdexcept> // For std::invalid_argument, std::out_of_range,  
std::runtime_error  
#include <cstdlib> // For EXIT_SUCCESS, EXIT_FAILURE  
  
//=====  
// 1. CONFIGURATION AND DATA DEFINITIONS  
//=====  
  
/**  
 * @def MAX_PERMUTATION_LENGTH  
 * @brief Hard-coded safety limit for the maximum permutation length.  
 *  
 * This prevents extreme memory usage or unbounded runtime by limiting how  
 * long individual permutations may be.  
 */  
constexpr std::uint64_t MAX_PERMUTATION_LENGTH = 26;  
  
//=====  
// 1.1 DOMINIC'S 98-SYMBOL ALPHABET  
//=====  
//  
// Index mapping (1-based, conceptual):  
// 1-26 : a-z  
// 27-52 : A-Z  
// 53-62 : 0-9  
// 63 : +  
// 64 : -  
// 65 : *  
// 66 : /  
// 67 : %  
// 68 : =  
// 69 : !  
// 70 : <  
// 71 : >  
// 72 : &  
// 73 : |
```

```

// 74      : ^
// 75      : ~
// 76      : ?
// 77      : :
// 78      : ;
// 79      : ,
// 80      : .
// 81      : (
// 82      : )
// 83      : [
// 84      : ]
// 85      : {
// 86      : }
// 87      : #
// 88      : "
// 89      : '
// 90      : bslash
// 91      : -
// 92      : @
// 93      : $
// 94      : `
// 95      : bslasht  (tab)
// 96      : ' ' (space)
// 97      : bslashn  (newline)
// 98      : bslash0  (null byte)
// 
// Note: This is stored as a raw char array (not a C-string). We always
// pass an explicit length when constructing std::string so that the '\0'
// symbol is treated as just another valid alphabet character.
static const char DOMINIC_ALPHABET_98[] = {

'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
'u', 'v', 'w', 'x', 'y', 'z',

'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
'U', 'V', 'W', 'X', 'Y', 'Z',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
'+', '-', '*', '/', '%', '=', '!', '<', '>', '&', '|', '^', '~', '?',
':', ';',
',', '.',
'(', ')', '[', ']', '{', '}', '#', '\'', '\"', '\\', '_', '@', '$', '^',
'\t', '\n', '\0'

};

//=====================================================================
// 2. ABSTRACT INTERFACE FOR OUTPUT (IOOutputSink)
//=====================================================================

/***
 * @class IOOutputSink
 * @brief An abstract interface for writing output.
 *
 * This decouples the core permutation logic from the output destination
 * (e.g., file, console, network socket). The core logic does not need to

```

```
* know *where* it is writing, only *how* to write via this interface.  
*/  
class IOutputSink {  
public:  
    virtual ~IOutputSink() = default;  
  
    /**  
     * @brief Write a buffer of bytes.  
     * @param buffer Pointer to the character buffer to write.  
     * @param size Number of bytes to write.  
     * @return true on success, false on failure.  
     */  
    virtual bool Write(const char* buffer, std::uint64_t size) = 0;  
  
    /**  
     * @brief Write a single character.  
     * @param c The character to write.  
     * @return true on success, false on failure.  
     */  
    virtual bool WriteChar(char c) = 0;  
};  
  
//=====  
// 3. FILE-BASED OUTPUT IMPLEMENTATION (FileSink)  
//=====  
  
/**  
 * @class FileSink  
 * @brief Concrete implementation of IOutputSink that writes to a file.  
 */  
class FileSink : public IOutputSink {  
public:  
    /**  
     * @brief Constructs a FileSink and opens the underlying file.  
     * @param filename Path to the output file.  
     *  
     * @throws std::runtime_error if the file cannot be opened.  
     */  
    explicit FileSink(const std::string& filename)  
        : m_file_stream(filename, std::ios::binary | std::ios::trunc)  
    {  
        if (!m_file_stream.is_open()) {  
            throw std::runtime_error("Failed to open output file: " + filename);  
        }  
    }  
  
    ~FileSink() override {  
        if (m_file_stream.is_open()) {  
            m_file_stream.close();  
        }  
    }  
  
    bool Write(const char* buffer, std::uint64_t size) override {  
        m_file_stream.write(buffer, static_cast<std::streamsize>(size));  
    }  
};
```

```
        return m_file_stream.good();
    }

    bool WriteChar(char c) override {
        m_file_stream.put(c);
        return m_file_stream.good();
    }

    FileSink(const FileSink&) = delete;
    FileSink& operator=(const FileSink&) = delete;

private:
    std::ofstream m_file_stream;
};

//=====
// 4. SAFE ARITHMETIC HELPERS
//=====

/***
 * @brief Safely multiply two std::uint64_t values with overflow detection.
 * @return true if result is valid, false if overflow would occur.
 */
static bool safe_uint64_mul(std::uint64_t a, std::uint64_t b, std::uint64_t& result) {
    if (a == 0 || b == 0) {
        result = 0;
        return true;
    }
    if (a > (std::numeric_limits<std::uint64_t>::max() / b)) {
        return false; // Overflow
    }
    result = a * b;
    return true;
}

/***
 * @brief Safely compute base^exp for std::uint64_t with overflow detection.
 * @param base The base value.
 * @param exp The exponent.
 * @param result Output parameter for the result if successful.
 * @return true on success, false on overflow.
 */
static bool safe_uint64_power(std::uint64_t base, std::uint64_t exp,
    std::uint64_t& result) {
    result = 1;
    for (std::uint64_t i = 0; i < exp; ++i) {
        std::uint64_t tmp = 0;
        if (!safe_uint64_mul(result, base, tmp)) {
            return false; // Overflow
        }
        result = tmp;
    }
    return true;
}
```

```
}

//=====
// 5. CORE PERMUTATION GENERATION LOGIC
//=====

/***
 * @brief Generate all permutations of a given length over the provided charset.
 *
 * Each permutation is written as a sequence of bytes of length @p length,
 * followed by a newline character ('\n'). Note that because the alphabet
 * may itself contain '\n' and '\0', the resulting file should be treated
 * as a binary file if using @dom98.
 *
 * @param length Length of permutations to generate (e.g., 3 for "aaa").
 * @param charset Character set to use for generation.
 * @param sink Output sink to receive the permutations.
 * @return true on success, false on I/O error or arithmetic overflow.
 */
static bool generate_permutations(
    std::uint64_t length,
    const std::string& charset,
    IOOutputSink& sink
) {
    const std::uint64_t charset_size = charset.size();

    if (charset_size == 0) {
        std::cerr << "Error: Charset size is zero.\n";
        return false;
    }

    std::uint64_t num_combinations = 0;
    if (!safe_uint64_power(charset_size, length, num_combinations)) {
        std::cerr << "Error: Overflow computing number of combinations for length "
        << length << ".\n";
        return false;
    }

    if (num_combinations == 0) {
        // Trivial case: nothing to generate.
        return true;
    }

    // Preallocate a buffer for one permutation line (without newline).
    std::string buffer(length, '\0');

    for (std::uint64_t index = 0; index < num_combinations; ++index) {
        std::uint64_t value = index;

        // Convert `index` into base-charset_size representation.
        for (std::int64_t pos = static_cast<std::int64_t>(length) - 1; pos >= 0; -pos) {
            const std::uint64_t digit = value % charset_size;
```

```
        value /= charset_size;
        buffer[static_cast<std::size_t>(pos)] =
charset[static_cast<std::size_t>(digit)];
    }

    if (!sink.Write(buffer.data(), length)) {
        std::cerr << "Error: Failed to write permutation to sink.\n";
        return false;
    }
    if (!sink.WriteChar('\n')) {
        std::cerr << "Error: Failed to write newline to sink.\n";
        return false;
    }
}

return true;
}

//=====
// 6. USAGE / HELP
//=====

/***
 * @brief Print usage information for this executable.
 */
static void print_usage(const char* program_name) {
    std::cerr << "Usage:\n"
        << "  " << program_name << " <charset[@dom98] <min_length>
<max_length>\n\n"
        << "Examples:\n"
        << "  " << program_name << " \"abc\" 1 3\n"
        << "      # Use explicit charset 'a', 'b', 'c' with lengths 1 to
3.\n\n"
        << "  " << program_name << " @dom98 1 4\n"
        << "      # Use Dominic's 98-symbol alphabet (built in) with lengths
1 to 4.\n";
}

//=====
// 7. MAIN ENTRY POINT
//=====

int main(int argc, char* argv[]) {
    // --- 1. Argument Validation ---
    if (argc != 4) {
        print_usage(argv[0]);
        return EXIT_FAILURE;
    }

    std::string charset;
    std::uint64_t min_length = 0;
    std::uint64_t max_length = 0;

    // --- 2. Parse Runtime Parameters (with C++ exceptions) ---
}
```

```
try {
    std::string charset_arg = argv[1];

    // Special mode: use Dominic's 98-symbol alphabet (size 98).
    // This includes printable ASCII, operators, space, tab, newline, and a
    // literal '\0' symbol as the 98th character.
    if (charset_arg == "@dom98") {
        charset.assign(
            DOMINIC_ALPHABET_98,
            DOMINIC_ALPHABET_98 + sizeof(DOMINIC_ALPHABET_98)
        );
    } else {
        // Original behaviour: use argv[1] literally as the charset string.
        charset = charset_arg;
    }

    // Use std::stoull for robust parsing of lengths.
    min_length = std::stoull(argv[2]);
    max_length = std::stoull(argv[3]);
}

catch (const std::invalid_argument&) {
    std::cerr << "Error: Invalid characters in length arguments. Must be
integers.\n";
    print_usage(argv[0]);
    return EXIT_FAILURE;
}

catch (const std::out_of_range&) {
    std::cerr << "Error: Length arguments are out of range for a 64-bit
integer.\n";
    print_usage(argv[0]);
    return EXIT_FAILURE;
}

// --- 3. Safety-Critical Parameter Validation ---

if (charset.empty()) {
    std::cerr << "Error: Charset cannot be empty.\n";
    return EXIT_FAILURE;
}

if (min_length == 0) {
    std::cerr << "Error: min_length must be at least 1.\n";
    return EXIT_FAILURE;
}

if (max_length < min_length) {
    std::cerr << "Error: max_length (" << max_length
                  << ") is less than min_length (" << min_length << ").\n";
    return EXIT_FAILURE;
}

if (max_length > MAX_PERMUTATION_LENGTH) {
    std::cerr << "Error: Requested max_length (" << max_length
                  << ") exceeds safety limit MAX_PERMUTATION_LENGTH ("
```

```

        << MAX_PERMUTATION_LENGTH << ").\n";
    return EXIT_FAILURE;
}

// --- 4. Generate permutations for each requested length ---

bool success = true;

for (std::uint64_t length = min_length; length <= max_length; ++length) {
    std::stringstream filename_builder;
    filename_builder << "permutations_len_" << length << ".txt";
    const std::string filename = filename_builder.str();

    std::cout << "Generating permutations of length " << length
        << " into file: " << filename << "....\n";

    try {
        FileSink sink(filename);

        if (!generate_permutations(length, charset, sink)) {
            std::cerr << "Error: Generation failed for length " << length <<
".\n";
            success = false;
            break;
        }
    }
    catch (const std::runtime_error& e) {
        std::cerr << "Error: " << e.what() << "\n";
        success = false;
        break;
    }
}

if (success) {
    std::cout << "All generations complete.\n";
    return EXIT_SUCCESS;
} else {
    std::cerr << "Generation failed and was halted.\n";
    return EXIT_FAILURE;
}
}
}

```

## [46] GenesisOS/system.php

- **Bytes:** 7299
- **Type:** text

```

<?php
/**
 * Text Reversion

```

```
* System Module
*/
/* =====
 * SYSTEM MODULE
 * ===== */
/***
 * Handle system API requests
 */
function handleSystemAPI() {
    global $config;
    $method = $_POST['method'] ?? $_GET['method'] ?? '';
    $params = $_POST['params'] ?? [];

    // Parse params if it's a JSON string
    if (is_string($params)) {
        $params = json_decode($params, true) ?? [];
    }

    switch ($method) {
        case 'getSystemInfo':
            return [
                'success' => true,
                'data' => [
                    'name' => $config['system']['name'],
                    'version' => $config['system']['version'],
                    'build' => $config['system']['build'],
                    'php_version' => PHP_VERSION,
                    'server' => $_SERVER['SERVER_SOFTWARE'] ?? 'Unknown',
                    'debug_mode' => $config['system']['debug'],
                    'uptime' => time() - $_SERVER['REQUEST_TIME_FLOAT'],
                    'use_local_filesystem' => $config['filesystem']['use_local'],
                    //  FIXED: Expose the entire security configuration block
                    to the frontend.
                    // This ensures that any security flags needed by the client
                    are available.
                    'security' => $config['security'] ?? []
                ]
            ];

        case 'updateConfig':
            // Only admins can update config
            if (!isAdmin()) {
                return ['success' => false, 'message' => 'Access denied'];
            }

            if (isset($params['key']) && isset($params['value'])) {
                $key = $params['key'];
                $value = $params['value'];

                // Update config (in a real system, would save to a file/database)
                $keys = explode('.', $key);
                $ref = &$config;
```

```
        foreach ($keys as $k) {
            if (!isset($ref[$k])) {
                return ['success' => false, 'message' => 'Invalid config
key'];
            }
            $ref = &$ref[$k];
        }

        $ref = $value;

        return ['success' => true, 'message' => 'Configuration updated'];
    }
    break;

case 'logError':
    global $errorLogFile;
    if (isset($params['message'])) {
        $details = $params['details'] ?? '';
        $entry = '[' . date('Y-m-d H:i:s') . "] JS {$params['message']}";
        if ($details) { $entry .= "\n$details"; }
        file_put_contents($errorLogFile, $entry . "\n", FILE_APPEND);
        return ['success' => true];
    }
    break;

case 'getLanguagePack':
    $lang = $params['lang'] ?? ($config['ui']['language'] ?? 'en');
    $pack = loadLanguagePack($lang);
    if (empty($pack) && $lang != 'en') {
        $pack = loadLanguagePack('en');
    }
    return ['success' => true, 'data' => $pack];

case 'runSelfTests':
    if (!isAdmin()) {
        return ['success' => false, 'message' => 'Access denied'];
    }

    // This captures any stray output to prevent corrupting the JSON
    response.
    ob_start();

    $tests = [];
    $tests['php_version_ok'] = version_compare(PHP_VERSION, '7.4', '>');
    $tests['user_dir_writable'] = is_writable($config['filesystem']
    ['user_dir']);
    $tests['error_log_writable'] = is_writable($config['filesystem']
    ['system_dir'] . '/error.log') || is_writable($config['filesystem']
    ['system_dir']);
    $tests['sessions_enabled'] = session_status() === PHP_SESSION_ACTIVE;

    // Clean the buffer before returning the result.
    ob_end_clean();
```

```
        return ['success' => true, 'data' => $tests];
    }

    return ['success' => false, 'message' => 'Invalid method or parameters'];
}

/* =====
 * SYSTEM INITIALIZATION
 * ===== */
function initializeSystem() {
    global $config;

    if ($config['filesystem']['use_local']) {
        $readmePath = $config['filesystem']['root_dir'] . '/README.txt';
        $userDBFile = $config['filesystem']['system_dir'] . '/users.json';
        $appSubmissionsFile = $config['filesystem']['system_dir'] .
            '/app_submissions.json';
        $errorLogFile = $config['filesystem']['system_dir'] . '/error.log';

        // Default README
        if (!file_exists($readmePath)) {
            $readme = "Welcome to Genesis OS. Login with admin/pwd or
guest/pwdguest.";
            file_put_contents($readmePath, $readme);
        }

        // Default user database
        if (!file_exists($userDBFile)) {
            $defaultUsers = [
                [
                    'username' => 'admin',
                    'password' => password_hash('pwd', PASSWORD_DEFAULT),
                    'roles' => ['admin', 'developer'],
                    'name' => 'Administrator',
                    'quota' => 20 * 1024 * 1024,
                    'lastLogin' => null
                ],
                [
                    'username' => 'guest',
                    'password' => password_hash('pwdguest', PASSWORD_DEFAULT),
                    'roles' => ['user'],
                    'name' => 'Guest User',
                    'quota' => 10 * 1024 * 1024,
                    'lastLogin' => null
                ]
            ];
            file_put_contents($userDBFile, json_encode($defaultUsers,
JSON_PRETTY_PRINT));
        }

        // Default submissions file
        if (!file_exists($appSubmissionsFile)) {
            file_put_contents($appSubmissionsFile, json_encode([]),

```

```
JSON_PRETTY_PRINT));
}

// Default error log
if (!file_exists($errorLogFile)) {
    file_put_contents($errorLogFile, "");
}

// User home directories
$users = loadUserDB();
foreach ($users as $u) {
    $home = $config['filesystem']['user_dir'] . '/' . $u['username'];
    $desktop = $home . '/Desktop';
    $documents = $home . '/Documents';

    if (!is_dir($home)) mkdir($home, 0755, true);
    if (!is_dir($desktop)) mkdir($desktop, 0755, true);
    if (!is_dir($documents)) mkdir($documents, 0755, true);
}
}

/***
 * Load a language pack
 */
function loadLanguagePack(string $code): array {
    global $config;
    $languageDir = $config['filesystem']['system_dir'] . '/lang';
    if(!is_dir($languageDir)) mkdir($languageDir, 0755, true);

    $file = "$languageDir/lang_{$code}.json";
    if (!file_exists($file)) {
        if ($code === 'en') { // Create default english pack if not exists
            $defaultLang = [
                'login_heading' => 'Genesis OS Login', 'username' => 'Username',
                'password' => 'Password',
                'login_button' => 'Log In', 'login_missing' => 'Username and
password are required.'
            ];
            file_put_contents($file, json_encode($defaultLang,
JSON_PRETTY_PRINT));
            return $defaultLang;
        }
        return [];
    }
    $json = json_decode(file_get_contents($file), true);
    return is_array($json) ? $json : [];
}
```

- **Bytes:** 282
- **Type:** text

A spanning basis of all integers including zero, is formable with the use of a set of two or more prime numbers. Where a spanning basis is able to generate (efficiently or otherwise): the state of acceptance (alpha). Where alpha is the attainment of all hope. Also known as, Heaven.

## [48] GenesisOS/tr-api.php

- **Bytes:** 3276
- **Type:** text

```
<?php
require_once 'auth.php';
require_once 'VersionManager.php';

// Authenticate every API request
if (!isLoggedIn()) {
    header('HTTP/1.1 403 Forbidden');
    echo json_encode(['success' => false, 'message' => 'Authentication required.']);
    exit;
}

// All responses are JSON
header('Content-Type: application/json');

// Get POST data
$action = $_POST['action'] ?? '';
$params = isset($_POST['params']) ? json_decode($_POST['params'], true) : [];

$response = ['success' => false, 'message' => 'Invalid action'];

try {
    // Define paths
    // Assumes the workbench is adjacent to the live platform directory
    $live_tr_path = realpath(__DIR__ . '/../tr');
    $versions_dir = __DIR__ . '/versions';

    if (!$live_tr_path) {
        throw new Exception('FATAL: Live BEP directory "berto-tools-platform" not found adjacent to the workbench.');
    }

    // ☑ FIXED: Moved manager instantiation inside the try block.
    // This ensures that if the constructor fails, the error is caught
    // and formatted as a proper JSON response.
    $manager = new VersionManager($live_tr_path, $versions_dir);
}
```

```
switch ($action) {
    case 'listVersions':
        $response = ['success' => true, 'data' => $manager->listVersions()];
        break;

    case 'createVersion':
        $versionName = $manager->createVersion();
        $response = ['success' => true, 'message' => "Version '$versionName' created successfully."];
        break;

    case 'listVersionFiles':
        $version = $params['version'] ?? null;
        $response = ['success' => true, 'data' => $manager->listVersionFiles($version)];
        break;

    case 'readFile':
        $version = $params['version'] ?? null;
        $file = $params['file'] ?? null;
        $response = ['success' => true, 'data' => $manager->readFile($version, $file)];
        break;

    case 'writeFile':
        $version = $params['version'] ?? null;
        $file = $params['file'] ?? null;
        $content = $params['content'] ?? '';
        $manager->writeFile($version, $file, $content);
        $response = ['success' => true, 'message' => 'File saved successfully.'];
        break;

    case 'runTests':
        $version = $params['version'] ?? null;
        $response = ['success' => true, 'data' => $manager->runTests($version)];
        break;

    case 'deployVersion':
        $version = $params['version'] ?? null;
        $manager->deployVersion($version);
        $response = ['success' => true, 'message' => "Version '$version' has been deployed to the live environment."];
        break;

    case 'deleteVersion':
        $version = $params['version'] ?? null;
        $manager->deleteVersion($version);
        $response = ['success' => true, 'message' => "Version '$version' has been deleted."];
        break;
}

} catch (Exception $e) {
```

```
// Now, any error (including from the constructor) will be caught here.  
$response = ['success' => false, 'message' => $e->getMessage()];  
}  
  
echo json_encode($response);
```

## [49] GenesisOS/tr-auth.php

- **Bytes:** 1654
- **Type:** text

```
<?php  
session_start();  
  
// --- CONFIGURATION ---  
// The username for the workbench administrator.  
define('ADMIN_USERNAME', 'tr_admin');  
  
// IMPORTANT: Replace this placeholder with a securely generated password hash.  
// To generate a hash, run this PHP code once:  
//echo password_hash('00EITA00', PASSWORD_DEFAULT);  
define('ADMIN_PASSWORD_HASH',  
'$2y$10$zLx57y0qXPte0k9oaISIXu.JgDkBSayk2TL0foJry0aGtGrtRfmLy');  
// --- END CONFIGURATION ---  
  
/**  
 * Checks if the current user is authenticated.  
 * @return bool True if logged in, false otherwise.  
 */  
function isLoggedIn(): bool {  
    return isset($_SESSION['is_logged_in']) && $_SESSION['is_logged_in'] === true;  
}  
  
/**  
 * Attempts to log a user in.  
 * @param string $username The submitted username.  
 * @param string $password The submitted password.  
 * @return bool True on successful login, false otherwise.  
 */  
function login(string $username, string $password): bool {  
    if ($username === ADMIN_USERNAME && password_verify($password,  
ADMIN_PASSWORD_HASH)) {  
        // Regenerate session ID to prevent session fixation attacks.  
        session_regenerate_id(true);  
        $_SESSION['is_logged_in'] = true;  
        return true;  
    }  
    return false;  
}  
  
/**
```

```

 * Logs the current user out by destroying the session.
 */
function logout(): void {
    $_SESSION = [];
    if (ini_get("session.use_cookies")) {
        $params = session_get_cookie_params();
        setcookie(session_name(), '', time() - 42000,
            $params["path"], $params["domain"],
            $params["secure"], $params["httponly"])
    }
    session_destroy();
}

```

## [50] GenesisOS/tr-generate\_hash.php

- **Bytes:** 109
- **Type:** text

```

<?php
// Replace with the strong password you want to use.
echo password_hash('pwd', PASSWORD_DEFAULT);
?>

```

## [51] GenesisOS/tr-index.php

- **Bytes:** 4723
- **Type:** text

```

<?php
require_once 'auth.php';

if (!isLoggedIn()) {
    header('Location: login.php');
    exit;
}

if (isset($_GET['action']) && $_GET['action'] === 'logout') {
    logout();
    header('Location: login.php');
    exit;
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TR Admin Workbench</title>

```

```
<link rel="stylesheet" href="assets/style.css">
</head>
<body>
    <div class="main-container">
        <aside class="sidebar">
            <div class="sidebar-header">
                <h2>TR Workbench</h2>
                <a href="index.php?action=logout">Logout</a>
            </div>
            <div class="sidebar-content">
                <h3>Versions</h3>
                <button id="create-version-btn" class="button-primary">Create New Version</button>
                <ul id="versions-list">
                    </ul>
            </div>
        </aside>

        <main class="main-content">
            <div class="content-panel" id="file-browser-panel">
                <div class="panel-header">
                    <h3 id="file-browser-heading">Select a Version</h3>
                </div>
                <div class="panel-body" id="file-tree">
                    </div>
            </div>

            <div class="content-panel" id="editor-panel">
                <div class="panel-header">
                    <h3 id="editor-heading">Editor</h3>
                    <div id="editor-actions" style="display: none;">
                        <button id="save-file-btn">Save Changes</button>
                    </div>
                </div>
                <div class="panel-body">
                    <div id="editor-container"></div>
                    <div id="editor-placeholder">
                        <p>Select a file from the browser to view or edit its contents.</p>
                    </div>
                </div>
            </div>

            <div class="content-panel" id="actions-panel">
                <div class="panel-header">
                    <h3>Actions & Testing</h3>
                </div>
                <div class="panel-body" id="actions-body">
                    <div id="actions-placeholder"><p>Select a version to see available actions.</p></div>
                    <div id="actions-content" style="display:none;">
                        <h4>Run Automated Tests</h4>
                        <p>Verify the integrity and stability of this version before deployment.</p>
                    </div>
                </div>
            </div>
        </main>
    </div>
</body>
```

```
<button id="run-tests-btn">Run Tests</button>
<pre id="test-results-output"></pre>

<hr>

<h4>Deployment</h4>
<p>Deploy this version to the live environment. The current live version will be backed up automatically.</p>
<button id="deploy-btn" class="button-danger" disabled>Deploy Version</button>

<hr>

<h4>Other Actions</h4>
<button id="download-btn">Download (.zip)</button>
<button id="delete-btn" class="button-danger">Delete Version</button>

</div>
</div>
</div>
</main>
</div>

<div id="modal" class="modal-backdrop" style="display:none;">
<div class="modal-content">
<h3 id="modal-title">Notice</h3>
<p id="modal-message"></p>
<div class="modal-actions">
<button id="modal-confirm-btn">OK</button>
<button id="modal-cancel-btn" style="display:none;">Cancel</button>
</div>
</div>
</div>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.15/codemirror.min.js">
</script>
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.15/codemirror.min.css">
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.15/mode/php/php.js">
</script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.15/mode/xml/xml.js">
</script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.15/mode/javascript/javascript.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.15/mode/css/css.js">
</script>
<script>
```

```
src="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.65.15/mode/htmlmixed/htmlmixed.js"></script>

<script src="assets/script.js"></script>
</body>
</html>
```

## [52] GenesisOS/tr-login.php

- **Bytes:** 1414
- **Type:** text

```
<?php
require_once 'auth.php';

$error = '';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'] ?? '';
    $password = $_POST['password'] ?? '';
    if (login($username, $password)) {
        header('Location: index.php');
        exit;
    } else {
        $error = 'Invalid username or password.';
    }
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TR Admin Workbench - Login</title>
    <link rel="stylesheet" href="assets/style.css">
</head>
<body class="login-page">
    <div class="login-container">
        <h1>TR Admin Workbench</h1>
        <p>Please log in to continue.</p>
        <form method="POST" action="login.php">
            <div class="form-group">
                <label for="username">Username</label>
                <input type="text" id="username" name="username" required>
            </div>
            <div class="form-group">
                <label for="password">Password</label>
                <input type="password" id="password" name="password" required>
            </div>
            <?php if ($error): ?>
                <p class="error-message"><?php echo $error; ?></p>
            <?php endif; ?>
        </form>
    </div>
</body>
```

```
<button type="submit">Log In</button>
</form>
</div>
</body>
</html>
```

## [53] GenesisOS/tr-programming.js

- **Bytes:** 2439
- **Type:** text

```
// extend/tr-programming.js – TR programming quick manual (classic, v1.0.3)
// Works without ESM. Provides :docs and contributes a help line.

(function () {
    function register(tr) {
        const manual = [
            "GENESIS OS – TR PROGRAMMING MANUAL (concise)",
            "",
            "OVERVIEW",
            "  TR Terminal is an extensible REPL. Packages (.js) live in /extend/",
            "  alongside tr-terminal-config.json.",
            "",
            "FILES & LAYOUT",
            "  /apps/tr-terminal.json",
            "  /apps/tr-terminal.js",
            "  /apps/extend/tr-terminal-config.json",
            "  /apps/extend/tr-programming.js  (this package)",
            "",
            "CONFIG SCHEMA (tr-terminal-config.json)",
            "  {",
            "    version: \"1.0.0\",
            "    installed: {",
            "      name: { filename, enabled, type, hash, installedAt, updatedAt }",
            "    },
            "    order: [\"name1\", \"name2\", ...]",
            "  }",
            "",
            "WRITING A PACKAGE (classic)",
            "  window.trTerminalPackage = {",
            "    register(tr){ tr.registerCommand('hello', (argv,
api)=>api.print('hi')); }",
            "  };",
            "",
            "PACKAGE API",
            "  tr.registerCommand(name, fn)      # add :name",
            "  tr.unregisterCommand(name)       # remove",
            "  tr.api.print(text, cls?)        # ok|err|muted",
            "  tr.api.getState()              # { admin, version, config }",
            "  tr.api.getConfig(),
            "  tr.api.setConfig(cb)",
```

```

    "  tr.api.saveConfig(),
    "  tr.api.fetchSelf(file)",
    "  tr.api.addAlias(alias, target)",
    "",
    "ADMIN WORKFLOW",
    "  :admin on • :install • :enable/:disable <name> • :uninstall <name> •
:reload",
    "  :config export|import|reset",
    "",
    "QUICK TEST",
    "  :docs      # show this manual",
    "",
    "END."
].join("\n"); // ← real newlines

tr.registerCommand("docs", (_argv, api) => api.print(manual));
tr.api.addAlias("manual", "docs");

try {
  const g = (typeof window !== "undefined" ? window : globalThis);
  if (!g.__trHelpAdditions) g.__trHelpAdditions = [];
  const line = "  :docs           Show the TR programming manual
(from tr-programming)";
  if (!g.__trHelpAdditions.includes(line)) g.__trHelpAdditions.push(line);
} catch (_) {}

tr.api.print("Tip: :docs – TR programming manual (from tr-programming)",
"muted");
}

window.trTerminalPackage = { register };
})();

```

## [54] GenesisOS/tr-script.js

- **Bytes:** 13391
- **Type:** text

```

document.addEventListener('DOMContentLoaded', () => {
  // --- STATE MANAGEMENT ---
  const state = {
    versions: [],
    selectedVersion: null,
    selectedFile: null,
    isDirty: false, // Tracks if editor has unsaved changes
    editor: null,
  };

  // --- DOM SELECTORS ---
  const versionsList = document.getElementById('versions-list');

```

```
const createVersionBtn = document.getElementById('create-version-btn');
const fileTreeContainer = document.getElementById('file-tree');
const fileBrowserHeading = document.getElementById('file-browser-heading');
const editorContainer = document.getElementById('editor-container');
const editorPlaceholder = document.getElementById('editor-placeholder');
const editorActions = document.getElementById('editor-actions');
const saveFileBtn = document.getElementById('save-file-btn');
const editorHeading = document.getElementById('editor-heading');

const actionsPlaceholder = document.getElementById('actions-placeholder');
const actionsContent = document.getElementById('actions-content');
const runTestsBtn = document.getElementById('run-tests-btn');
const testResultsOutput = document.getElementById('test-results-output');
const deployBtn = document.getElementById('deploy-btn');
const downloadBtn = document.getElementById('download-btn');
const deleteBtn = document.getElementById('delete-btn');

// --- API HELPER ---
async function apiCall(action, params = {}) {
    const formData = new FormData();
    formData.append('action', action);
    formData.append('params', JSON.stringify(params));

    try {
        const response = await fetch('api.php', {
            method: 'POST',
            body: formData
        });

        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }

        const result = await response.json();
        if (!result.success) {
            throw new Error(result.message);
        }
        return result.data;
    } catch (error) {
        showModal('API Error', `An error occurred: ${error.message}`);
        throw error; // Re-throw to stop promise chains
    }
}

// --- MODAL/NOTIFICATION ---
function showModal(title, message, options = {}) {
    const modal = document.getElementById('modal');
    document.getElementById('modal-title').textContent = title;
    document.getElementById('modal-message').innerHTML = message;

    const confirmBtn = document.getElementById('modal-confirm-btn');
    const cancelBtn = document.getElementById('modal-cancel-btn');

    modal.style.display = 'flex';
}
```

```
        return new Promise((resolve) => {
            confirmBtn.onclick = () => { modal.style.display = 'none';
            resolve(true); };
            cancelBtn.onclick = () => { modal.style.display = 'none';
            resolve(false); };

            if (options.confirmText) confirmBtn.textContent = options.confirmText;
            if (options.showCancel) {
                cancelBtn.style.display = 'inline-block';
            } else {
                cancelBtn.style.display = 'none';
            }
        });
    }

// --- EDITOR ---
function initializeEditor() {
    state.editor = CodeMirror(editorContainer, {
        lineNumbers: true,
        mode: 'php',
        theme: 'default', // Using CSS variables, a custom theme isn't
strictly needed
    });
    state.editor.on('change', () => {
        if (!state.isDirty) {
            state.isDirty = true;
            editorHeading.textContent += ' *';
        }
    });
}

// --- RENDER FUNCTIONS ---
function renderVersions() {
    versionsList.innerHTML = '';
    state.versions.forEach(version => {
        const li = document.createElement('li');
        li.dataset.version = version;
        li.textContent = version;
        if (version === state.selectedVersion) {
            li.classList.add('active');
        }
        li.addEventListener('click', () => selectVersion(version));
        versionsList.appendChild(li);
    });
}

function renderFileTree(files) {
    fileTreeContainer.innerHTML = '';
    // Simple, non-nested tree for now
    files.sort((a, b) => {
        if (a.type === 'dir' && b.type === 'file') return -1;
        if (a.type === 'file' && b.type === 'dir') return 1;
        return a.path.localeCompare(b.path);
    });
}
```

```
});

files.forEach(file => {
  const item = document.createElement('div');
  item.className = 'file-tree-item';
  item.dataset.path = file.path;
  item.dataset.type = file.type;

  const indent = (file.path.split('/').length - 1) * 15;
  item.style.paddingLeft = `${indent + 8}px`;

  const icon = document.createElement('span');
  icon.className = 'icon';
  icon.textContent = file.type === 'dir' ? '📁' : '📄';

  const name = document.createElement('span');
  name.textContent = file.path.split('/').pop();

  item.appendChild(icon);
  item.appendChild(name);

  if (file.type === 'file') {
    item.addEventListener('click', () => selectFile(file.path));
  }

  if(state.selectedFile === file.path){
    item.classList.add('active');
  }

  fileTreeContainer.appendChild(item);
});

function updateActionsPanel() {
  if (state.selectedVersion) {
    actionsPlaceholder.style.display = 'none';
    actionsContent.style.display = 'block';
    testResultsOutput.innerHTML = '';
    deployBtn.disabled = true;
  } else {
    actionsPlaceholder.style.display = 'block';
    actionsContent.style.display = 'none';
  }
}

// --- ACTION HANDLERS ---
async function loadVersions() {
  state.versions = await apiCall('listVersions');
  renderVersions();
}

async function selectVersion(version) {
  if (state.isDirty) {
    const abandon = await showModal('Unsaved Changes', 'You have unsaved
```

```
changes. Do you want to discard them and switch versions?', {showCancel: true});
    if (!abandon) return;
}

state.selectedVersion = version;
state.selectedFile = null;
state.isDirty = false;

// Update UI
renderVersions();
fileBrowserHeading.textContent = `Version: ${version}`;
editorPlaceholder.style.display = 'block';
editorContainer.style.display = 'none';
editorActions.style.display = 'none';
editorHeading.textContent = 'Editor';

updateActionsPanel();

// Load files
const files = await apiCall('listVersionFiles', { version });
renderFileTree(files);
}

async function selectFile(filePath) {
    if (state.isDirty) {
        const abandon = await showModal('Unsaved Changes', 'You have unsaved
changes. Do you want to discard them and open a new file?', {showCancel: true});
        if (!abandon) return;
    }

    state.selectedFile = filePath;

    // Update file tree selection
    document.querySelectorAll('.file-tree-item').forEach(el =>
el.classList.remove('active'));
    document.querySelector(`.file-tree-item[data-
path="${filePath}"]`).classList.add('active');

    // Load file content
    const content = await apiCall('readFile', { version:
state.selectedVersion, file: filePath });

    editorPlaceholder.style.display = 'none';
    editorContainer.style.display = 'block';
    editorActions.style.display = 'block';

    state.editor.setValue(content);
    state.isDirty = false;
    editorHeading.textContent = `Editing: ${filePath}`;
}

// --- EVENT LISTENERS ---
createVersionBtn.addEventListener('click', async () => {
    try {
```

```
        await apiCall('createVersion');
        await loadVersions();
        showModal('Success', 'A new version has been created from the live
system.');
    } catch (error) {
        // Error is handled in apiCall
    }
});

saveFileBtn.addEventListener('click', async () => {
    if (!state.selectedVersion || !state.selectedFile) return;

    try {
        const content = state.editor.getValue();
        await apiCall('writeFile', {
            version: state.selectedVersion,
            file: state.selectedFile,
            content: content
        });
        state.isDirty = false;
        editorHeading.textContent = `Editing: ${state.selectedFile}`;
        showModal('Success', 'File saved successfully.');
    } catch (error) {
        // Error handled in apiCall
    }
});

runTestsBtn.addEventListener('click', async () => {
    if (!state.selectedVersion) return;
    testResultsOutput.textContent = 'Running tests...';
    deployBtn.disabled = true;

    try {
        const results = await apiCall('runTests', { version:
state.selectedVersion });
        let outputHtml = '';
        let allPassed = true;

        for (const [testName, result] of Object.entries(results)) {
            const statusClass = result.passed ? 'test-pass' : 'test-fail';
            const statusText = result.passed ? 'PASSED' : 'FAILED';
            if (!result.passed) allPassed = false;

            outputHtml += `<div><strong>${testName}</strong> <span
class="${statusClass}">${statusText}</span></div>`;
            outputHtml += `<small>${result.details}</small><br><br>`;
        }
        testResultsOutput.innerHTML = outputHtml;

        if (allPassed) {
            deployBtn.disabled = false;
            showModal('Tests Passed', 'All tests passed successfully. You can
now deploy this version.');
        } else {
    
```

```
        showModal('Tests Failed', 'Some tests failed. Please review the
output before attempting to deploy.');
    }
} catch (error) {
    testResultsOutput.textContent = `Error running tests:
${error.message}`;
}
});

deployBtn.addEventListener('click', async () => {
    if (!state.selectedVersion || deployBtn.disabled) return;

    const confirmed = await showModal(
        'Confirm Deployment',
        `Are you sure you want to deploy version
<strong>${state.selectedVersion}</strong> to the live site? The current site will
be backed up and replaced.`,
        { showCancel: true, confirmText: 'Deploy' }
    );

    if (confirmed) {
        try {
            await apiCall('deployVersion', { version: state.selectedVersion
});
            showModal('Deployment Successful', `Version
${state.selectedVersion} is now live.`);
            // Reset state
            state.selectedVersion = null;
            await loadVersions();
            renderVersions();
            fileTreeContainer.innerHTML = '';
            fileBrowserHeading.textContent = 'Select a Version';
            updateActionsPanel();
        } catch (error) {
            // Error handled in apiCall
        }
    }
});
};

downloadBtn.addEventListener('click', () => {
    if (!state.selectedVersion) return;
    // This is a direct link, not an API call, to trigger browser download.
    window.location.href = `VersionManager.php?
action=download&version=${state.selectedVersion}`;
});

deleteBtn.addEventListener('click', async () => {
    if (!state.selectedVersion) return;

    const confirmed = await showModal(
        'Confirm Deletion',
        `Are you sure you want to permanently delete version
<strong>${state.selectedVersion}</strong>? This action cannot be undone.`,
        { showCancel: true, confirmText: 'Delete' }
    );
});
```

```

    );

    if (confirmed) {
        try {
            await apiCall('deleteVersion', { version: state.selectedVersion
        });
            showModal('Success', `Version ${state.selectedVersion} was
        deleted.`);
            // Reset state
            state.selectedVersion = null;
            await loadVersions();
            renderVersions();
            fileTreeContainer.innerHTML = '';
            fileBrowserHeading.textContent = 'Select a Version';
            updateActionsPanel();
        } catch (error) {
            // Error handled in apiCall
        }
    }
});

// --- INITIALIZATION ---
function init() {
    initializeEditor();
    loadVersions();
}

init();
});

```

## [55] GenesisOS/tr-style.css

- **Bytes:** 5214
- **Type:** text

```

:root {
    --bg-dark: #1e1e1e;
    --bg-medium: #2a2a2a;
    --bg-light: #3c3c3c;
    --text-primary: #d4d4d4;
    --text-secondary: #a0a0a0;
    --accent-blue: #0e639c;
    --accent-green: #28a745;
    --accent-red: #dc3545;
    --border-color: #4a4a4a;
}

* { box-sizing: border-box; margin: 0; padding: 0; }

body {

```

```
font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
background-color: var(--bg-dark);
color: var(--text-primary);
font-size: 14px;
line-height: 1.6;
}

.login-page {
display: flex;
justify-content: center;
align-items: center;
height: 100vh;
}

.login-container {
background: var(--bg-medium);
padding: 40px;
border-radius: 8px;
border: 1px solid var(--border-color);
width: 100%;
max-width: 400px;
}

.login-container h1 { margin-bottom: 10px; }
.login-container p { margin-bottom: 20px; color: var(--text-secondary); }

.form-group { margin-bottom: 20px; }
.form-group label { display: block; margin-bottom: 5px; }
.form-group input {
width: 100%;
padding: 10px;
background: var(--bg-dark);
border: 1px solid var(--border-color);
border-radius: 4px;
color: var(--text-primary);
}

button, .button-primary, .button-danger {
padding: 10px 15px;
border: 1px solid var(--border-color);
border-radius: 4px;
background: var(--bg-light);
color: var(--text-primary);
cursor: pointer;
transition: background-color 0.2s;
}
button:hover { background: #454545; }

.button-primary { background: var(--accent-blue); border-color: var(--accent-blue); }
.button-primary:hover { background: #187bcd; }
.button-danger { background: var(--accent-red); border-color: var(--accent-red); color: white; }
```

```
.button-danger:hover { background: #e44d5a; }

.error-message { color: var(--accent-red); }

.main-container { display: flex; height: 100vh; }
.sidebar { width: 280px; background: var(--bg-medium); border-right: 1px solid var(--border-color); display: flex; flex-direction: column; }
.sidebar-header { padding: 20px; border-bottom: 1px solid var(--border-color); display: flex; justify-content: space-between; align-items: center; }
.sidebar-header a { color: var(--text-secondary); text-decoration: none; }
.sidebar-content { padding: 20px; flex-grow: 1; overflow-y: auto; }
.sidebar-content h3 { margin-bottom: 15px; }
.sidebar-content button { width: 100%; margin-bottom: 20px; }

#versions-list { list-style: none; }
#versions-list li {
    padding: 10px 15px;
    border-radius: 4px;
    cursor: pointer;
    margin-bottom: 5px;
    border: 1px solid transparent;
}
#versions-list li:hover { background: var(--bg-light); }
#versions-list li.active { background: var(--accent-blue); border-color: #187bcd; }
#versions-list li small { color: var(--text-secondary); }

.main-content { flex-grow: 1; display: grid; grid-template-columns: 300px 1fr; grid-template-rows: 1fr auto; gap: 1px; background-color: var(--border-color); }

.content-panel { background: var(--bg-dark); display: flex; flex-direction: column; }
#file-browser-panel { grid-row: 1 / 3; }
#editor-panel { grid-column: 2 / 3; }
#actions-panel { grid-column: 2 / 3; }

.panel-header { padding: 15px 20px; background: var(--bg-medium); border-bottom: 1px solid var(--border-color); display: flex; justify-content: space-between; align-items: center; }
.panel-body { padding: 20px; flex-grow: 1; overflow: auto; }
.panel-body hr { border: 0; border-top: 1px solid var(--border-color); margin: 20px 0; }

#file-tree { font-size: 13px; }
.file-tree-item { display: flex; align-items: center; padding: 4px 8px; border-radius: 3px; cursor: pointer; }
.file-tree-item:hover { background: var(--bg-medium); }
.file-tree-item.active { background: var(--bg-light); }
.file-tree-item .icon { margin-right: 8px; width: 16px; text-align: center; }

#editor-container { height: 100%; }
.CodeMirror { height: 100%; background: var(--bg-dark); font-size: 13px; }
.CodeMirror-gutters { background: var(--bg-dark) !important; border-right: 1px solid var(--border-color); }
```

```
.CodeMirror-linenumber { color: #858585; }

#editor-placeholder, #actions-placeholder { text-align: center; color: var(--text-secondary); padding-top: 50px; }

#actions-body button { margin-right: 10px; margin-bottom: 10px; }
#test-results-output {
    margin-top: 15px;
    background: #111;
    border: 1px solid var(--border-color);
    padding: 15px;
    font-family: monospace;
    white-space: pre-wrap;
    min-height: 100px;
    max-height: 200px;
    overflow-y: auto;
}
.test-pass { color: var(--accent-green); }
.test-fail { color: var(--accent-red); }

.modal-backdrop {
    position: fixed;
    top: 0; left: 0;
    width: 100%; height: 100%;
    background: rgba(0,0,0,0.6);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1000;
}
.modal-content {
    background: var(--bg-medium);
    padding: 30px;
    border-radius: 8px;
    min-width: 400px;
}
.modal-actions { margin-top: 20px; text-align: right; }
.modal-actions button { margin-left: 10px; }
```

## [56] GenesisOS/tr-terminal-config.json

- **Bytes:** 327
- **Type:** text

```
{
    "version": "1.0.0",
    "installed": {
        "tr-programming": {
            "filename": "extend/tr-programming.js",
            "enabled": true,
            "type": "ext",
```

```

    "hash": "00000000",
    "installedAt": "2025-01-01T00:00:00.000Z",
    "updatedAt": "2025-01-01T00:00:00.000Z"
  }
},
"order": [
  "tr-programming"
]
}

```

## [57] GenesisOS/tr-terminal.js

- **Bytes:** 28736
- **Type:** text

```

// tr-terminal.js - TR Terminal for Genesis OS (v1.0.0)
// Minimal centered REPL + admin-developer package system (patches/extensions).
// - Installs/Enables/Disables/Uninstalls .js packages
// - Manages tr-terminal-config.json (next to the app), also mirrors in
localStorage
// - Discovers packaged assets via ?api=apps&method=getAppAsset&appId=tr-
terminal&file=...
//
// Package format (ESM or classic IIFE both supported):
//   ESM:     export function register(tr) { tr.registerCommand('hello', (argv,
api)=>api.print('hi')); }
//   Classic: window.trTerminalPackage = { register(tr){ /* ... */ } }
//
// Provided API to packages:
//   tr.registerCommand(name, fn)
//   tr.unregisterCommand(name)
//   tr.api.print(msg, cls?)      // cls: "ok", "err", "muted"
//   tr.api.getState()           // { admin, version, config }
//   tr.api.getConfig() / setConfig(cb) / saveConfig()
//   tr.api.fetchSelf(file)      // fetch packaged asset
//   tr.api.addAlias(alias, targetCommandLine)
//
// Admin-only mutating ops are gated by admin mode.
//
// -----
export function initialize(gosApiProxy) {
  const appId = "tr-terminal";
  const PKG_DIR = "extend/";
  const CONFIG_FILE = PKG_DIR + "tr-terminal-config.json";

  function normalizePkgPath(file) {
    if (!file) return "";
    const clean = String(file).replace(/\//g, "").replace(/extend\//, "");
    return PKG_DIR + clean;
  }
}

// Admin mode ops
function adminOp(...args) {
  if (!gosApiProxy.isAdmin) return;
  const [cmd, ...args] = args;
  switch (cmd) {
    case "list":
      gosApiProxy.list();
      break;
    case "install":
      gosApiProxy.install(args);
      break;
    case "enable":
      gosApiProxy.enable(args);
      break;
    case "disable":
      gosApiProxy.disable(args);
      break;
    case "remove":
      gosApiProxy.remove(args);
      break;
    case "update":
      gosApiProxy.update(args);
      break;
    case "setConfig":
      gosApiProxy.setConfig(args);
      break;
    case "saveConfig":
      gosApiProxy.saveConfig();
      break;
    case "fetchSelf":
      gosApiProxy.fetchSelf();
      break;
    case "addAlias":
      gosApiProxy.addAlias(args);
      break;
    default:
      throw new Error(`Unknown admin command: ${cmd}`);
  }
}

// Public API
gosApiProxy.registerCommand("tr-terminal", {
  name: "tr-terminal",
  fn: (args) => {
    if (args.length === 0) {
      gosApiProxy.list();
    } else if (args[0] === "list") {
      gosApiProxy.list();
    } else if (args[0] === "install") {
      gosApiProxy.install(args);
    } else if (args[0] === "enable") {
      gosApiProxy.enable(args);
    } else if (args[0] === "disable") {
      gosApiProxy.disable(args);
    } else if (args[0] === "remove") {
      gosApiProxy.remove(args);
    } else if (args[0] === "update") {
      gosApiProxy.update(args);
    } else if (args[0] === "setConfig") {
      gosApiProxy.setConfig(args);
    } else if (args[0] === "saveConfig") {
      gosApiProxy.saveConfig();
    } else if (args[0] === "fetchSelf") {
      gosApiProxy.fetchSelf();
    } else if (args[0] === "addAlias") {
      gosApiProxy.addAlias(args);
    } else {
      gosApiProxy.list();
    }
  }
});

```

```
}
```

```
const container = gosApiProxy.window.getContainer();
const doc = container.ownerDocument;

// ----- Styles (single centered rectangle, topbar inside) -----
const style = doc.createElement("style");
style.textContent = [
  ":root",
  " --bg:#1e1e2e; --panel:#282a36; --text:#cdd6f4; --muted:#6c7086;",
  " --border:#313244; --focus:#f5c2e7; --ok:#70d392; --err:#e74c3c;",
  " --topbar-h:36px;",
  "}",
  ".root{font-family:'Fira Code','Cascadia Code','JetBrains Mono',monospace;
background:var(--bg);",
  " min-height:100%; display:flex; align-items:center; justify-content:center;
color:var(--text); padding:20px}",
  ".editor{width:94%; max-width:940px; height:76vh; background:var(--panel);",
  " border:2px solid var(--border); border-radius:8px; box-shadow:0 4px 30px
rgba(0,0,0,.3);",
  " display:grid; grid-template-rows:var(--topbar-h) 1fr; overflow:hidden;
position:relative}",
  ".editor:focus-within{border-color:var(--focus); box-shadow:0 0 0 3px
rgba(245,194,231,.3), 0 4px 30px rgba(0,0,0,.3)}",
  ".topbar{grid-row:1/2; display:flex; align-items:center; gap:8px; padding:6px
10px; border-bottom:1px solid var(--border)}",
  ".tag{display:inline-block; border:1px solid var(--border); padding:2px 6px;
border-radius:999px; color:#aeb3cf; margin-right:6px}",
  ".pill{display:inline-flex; align-items:center; gap:6px; padding:4px 8px;
border:1px solid var(--border); border-radius:999px; background:#1f2023;
color:#bfc3df; cursor:pointer}",
  ".shell{grid-row:2/3; display:flex; overflow:hidden}",
  ".left{min-width:45px; border-right:1px solid var(--border); color:var(--muted);",
  " padding:15px 10px 15px 15px; line-height:1.6; user-select:none;
overflow:auto; white-space:pre}",
  ".right{flex:1; display:flex; flex-direction:column; min-width:0}",
  ".log{flex:1; padding:12px 15px; overflow:auto; line-height:1.6; white-
space:pre-wrap}",
  ".input{border-top:1px solid var(--border); display:flex; gap:8px;
padding:10px}",
  ".cmd{flex:1; background:transparent; color:var(--text); border:none;
outline:none; font-size:16px}",
  ".cmd::placeholder{color:var(--muted)}",
  ".ok{color:var(--ok)} .err{color:var(--err)} .muted{color:var(--muted)}",
  ".panel{position:absolute; right:0; top:var(--topbar-h); height:calc(100% -
var(--topbar-h)); width:0; overflow:hidden;",
  " transition:width .25s ease; border-left:1px solid var(--border);
background:var(--panel); z-index:2}",
  ".panel.open{width:min(54%,520px)}",
  ".panel-inner{display:flex; flex-direction:column; height:100%}",
  ".tabs{display:flex; gap:6px; border-bottom:1px solid var(--border);
padding:8px}",
  ".tab{padding:6px 10px; border:1px solid var(--border); border-bottom:none;
```

```
border-radius:8px 8px 0 0; background:#222432; color:#cf3f4; cursor:pointer}" ,  
    ".tab.active{background:#1e2030}" ,  
    ".pane{flex:1; display:none; min-height:0}" ,  
    ".pane.active{display:flex; overflow:auto}" ,  
    ".mini{width:100%; background:#202233; color:#e9ecff; border:1px solid var(--  
border); border-radius:8px; padding:6px 8px; outline:none; font-size:13px}" ,  
    ".btn{background:#202124; color:#f1f1f1; border:1px solid var(--border);  
padding:7px 10px; border-radius:8px; cursor:pointer}" ,  
    ".btn:hover{filter:brightness(1.06)}" ,  
    ".list{flex:1; padding:8px; overflow:auto; font-size:13px}" ,  
    ".row{display:flex; gap:8px; align-items:center}" ,  
    ".hint{position:absolute; bottom:14px; left:50%; transform:translateX(-50%);  
color:var(--muted); font-size:12px; opacity:.8}"  
].join("\n");  
doc.head.appendChild(style);  
  
// ----- DOM -----  
container.innerHTML = "";  
const root = doc.createElement("div");  
root.className = "root";  
root.innerHTML = [  
    '<div class="editor" id="editor">',  
    '    <div class="topbar">',  
    '        <span class="tag" id="modeTag">MODE: user</span>' ,  
    '        <span class="tag" id="pkgTag">PKGS: 0</span>' ,  
    '        <span class="tag" id="verTag">TR v1.0.0</span>' ,  
    '        <span class="pill" id="panelBtn">Packages</span>' ,  
    '        <span class="pill" id="adminBtn">Admin: off</span>' ,  
    '    </div>' ,  
    '    <div class="shell">',  
    '        <div class="left" id="lines">1</div>' ,  
    '        <div class="right">',  
    '            <div class="log" id="log"></div>' ,  
    '            <div class="input">',  
    '                <input class="cmd" id="cmd" placeholder=":help -  
install/enable/disable/list/export/import packages; (:panel toggles package  
panel)" autocomplete="off"/>' ,  
    '                </div>' ,  
    '            </div>' ,  
    '            <div class="panel" id="panel">',  
    '                <div class="panel-inner">',  
    '                    <div class="tabs">',  
    '                        <div class="tab active" data-pane="pkgs">Packages</div>' ,  
    '                        <div class="tab" data-pane="config">Config</div>' ,  
    '                    </div>' ,  
    '                    <div class="pane active" id="pane-pkgs">',  
    '                        <div class="list" id="pkgList"></div>' ,  
    '                        <div class="row" style="padding:8px; border-top:1px solid var(--  
border)">',  
    '                            <input class="mini" id="fileName" placeholder="new-package.js  
(optional hint)" />' ,  
    '                            <button class="btn" id="installBtn">Install .js</button>' ,  
    '                            <button class="btn" id="reloadBtn">Reload enabled</button>' ,  
    '                        </div>' ,
```

```
'          </div>',
'          <div class="pane" id="pane-config">,
'            <div class="list" id="configView"></div>,
'            <div class="row" style="padding:8px; border-top:1px solid var(--border)">',
'              <button class="btn" id="exportCfg">Export config</button>,
'              <button class="btn" id="importCfg">Import config</button>,
'              <button class="btn" id="resetCfg">Reset (local)</button>,
'            </div>,
'          </div>,
'        </div>,
'      </div>,
'    </div>,
'  </div>,
'</div>',
'<div class="hint">Type <code>:help</code> for documentation</div>'  
].join("\n");
container.appendChild(root);

// ----- Small helpers -----
const $ = s => root.querySelector(s);
const on = (el, ev, fn) => el.addEventListener(ev, fn);
const linesEl = $("#lines");
const logEl = $("#log");
const cmdEl = $("#cmd");
const panel = $("#panel");
const pkgList = $("#pkgList");
const configView = $("#configView");

let lineCount = 1;
const print = (msg, cls="") => {
  const div = doc.createElement("div");
  if (cls) div.className = cls;
  div.textContent = msg;
  logEl.appendChild(div);
  logEl.scrollTop = logEl.scrollHeight;
  lineCount += String(msg).split("\n").length;
  linesEl.textContent = Array.from({length: lineCount}, (_, i) =>
i+1).join("\n");
};

const setTag = (id, text) => { const el = $("#" + id); if (el) el.textContent = text; };

// ----- Admin gating -----
const CTRL = { admin: false };
function requireAdmin() {
  if (!CTRL.admin) { print("admin-developer required. Use :admin on or click Admin button.", "err"); return false; }
  return true;
}

// appId already defined globally elsewhere as "tr-terminal"

async function fetchSelf(file) {
```

```
// 1) Compute the *directory* of the current document (e.g. "/tr/")
const path = location.pathname || "/";
const baseDir = path.endsWith("/")
? path
: path.replace(/\/[^\/]*$/, "/"); // strip last segment (e.g.
"sandbox.html")

// 2) Always target that dir's index.php (e.g. "/tr/index.php")
const basePath = baseDir + "index.php";

// 3) Preserve slashes in the file param
const fileParam = encodeURI(file); // encodes spaces, NOT '/'

// 4) Build a strictly relative, same-origin URL (no protocol/host)
const url = `${basePath}?${Date.now()}`;

api=apps&method=getAppAsset&appId=${encodeURIComponent(appId)}&file=${fileParam}&v
=${Date.now()}`;

const res = await fetch(url, {
  cache: "no-store",
  credentials: "same-origin",
  redirect: "follow"
});

if (!res.ok) throw new Error(`asset fetch failed: ${res.status}`);
const text = await res.text();
if (!text.trim()) throw new Error("empty response");
return text;
}

// ----- Config (disk-like via import/export + local mirror) -----
const LS_KEY = "tr-terminal.config";
const DEFAULT_CFG = {
  version: "1.0.0",
  installed: { /* name: { filename, enabled, type: 'patch'|'ext', hash,
installedAt, updatedAt } */ },
  order: []
};

let CONFIG = loadConfigLocal();

// Try to merge packaged config if present
(async () => {
  try {
    const cfgTxt = await fetchSelf("extend/tr-terminal-config.json");
    const packaged = JSON.parse(cfgTxt);
    mergeConfig(packaged);
    saveConfigLocal();
    print(`Loaded config from extend/tr-terminal-config.json`, "muted");
  } catch {
    print(`No packaged config in extend/tr-terminal-config.json (using local
mirror)`, "muted");
  }
})
```

```
refreshBadges();
renderPkgList();
renderConfigView();
})();

function loadConfigLocal() {
  try {
    const raw = localStorage.getItem(LS_KEY);
    if (!raw) return structuredClone(DEFAULT_CFG);
    const obj = JSON.parse(raw);
    return { ...structuredClone(DEFAULT_CFG), ...obj, installed: obj.installed
    || {}, order: obj.order || [] };
  } catch (_) { return structuredClone(DEFAULT_CFG); }
}

function saveConfigLocal() {
  localStorage.setItem(LS_KEY, JSON.stringify(CONFIG));
}

function mergeConfig(incoming) {
  if (!incoming || typeof incoming !== "object") return;
  CONFIG.version = CONFIG.version || incoming.version || "1.0.0";
  CONFIG.order = Array.isArray(CONFIG.order) ? CONFIG.order : [];
  const inc = incoming.installed || {};
  for (const name of Object.keys(inc)) {
    if (!CONFIG.installed[name]) {
      CONFIG.installed[name] = inc[name];
      CONFIG.order.push(name);
    }
  }
  // keep existing enabled flags as-is
}

function download(name, content, mime="application/json") {
  const blob = new Blob([content], { type: mime });
  const a = doc.createElement("a");
  a.href = URL.createObjectURL(blob);
  a.download = name;
  doc.body.appendChild(a);
  a.click();
  setTimeout(() => { URL.revokeObjectURL(a.href); a.remove(); }, 0);
}

function refreshBadges() {
  setTag("pkgTag", `PKGS: ${Object.keys(CONFIG.installed).length}`);
  setTag("modeTag", `MODE: ${CTRL.admin ? "admin" : "user"}`);
}

// ----- Package registry / REPL commands -----
const registry = {
  commands: Object.create(null),
  aliases: Object.create(null)
};

function registerCommand(name, fn) {
  if (!name || typeof fn !== "function") return false;
```

```
registry.commands[name] = fn;
return true;
}
function unregisterCommand(name) {
  delete registry.commands[name];
}
function addAlias(alias, target) {
  if (!alias || !target) return false;
  registry.aliases[alias] = target;
  return true;
}

// Built-in commands
registerCommand("echo", (argv, api) => api.print(argv.join(" ")));
registerCommand("help", (argv, api) => api.print(coreHelp));
registerCommand("state", (argv, api) => api.print(JSON.stringify({ admin: CTRL.admin, version: "1.0.0" }, null, 2)));
registerCommand("version", (argv, api) => api.print("TR Terminal v1.0.0"));

const coreHelp = [
  "TR TERMINAL - core commands",
  "  :help                      Show this help.",
  "  :admin on|off               Toggle admin developer mode.",
  "  :panel                     Toggle Packages panel.",
  "  :list                      List installed packages.",
  "  :install                   Pick a .js file (admin).",
  "  :enable <name>             Enable a package (admin).",
  "  :disable <name>            Disable a package (admin).",
  "  :uninstall <name>          Remove a package (admin).",
  "  :reload                    Reload all enabled packages.",
  "  :config export             Download tr-terminal-config.json.,
  "  :config import             Load config from file (admin).",
  "  :config reset              Reset local config (admin).",
  "",
  "Packages can register REPL commands; try :help again after install.",
].join("\n");

function apiObject() {
  return {
    print,
    getState: () => ({ admin: CTRL.admin, version: "1.0.0", config: CONFIG }),
    getConfig: () => JSON.parse(JSON.stringify(CONFIG)),
    setConfig: (cb) => { if (typeof cb === "function") { const next =
      cb(JSON.parse(JSON.stringify(CONFIG))); if (next) CONFIG = next; } },
    saveConfig: () => { saveConfigLocal(); renderPkgList(); renderConfigView();
    refreshBadges(); },
    fetchSelf: fetchSelf,
    addAlias
  };
}

const tr = {
  registerCommand,
  unregisterCommand,
```

```
    api: apiObject()
};

async function evalPackageText(name, codeText) {
    // 1) Try ESM via blob url
    try {
        const url = URL.createObjectURL(new Blob([codeText], { type: "text/javascript" }));
        try {
            const mod = await import(/* @vite-ignore */ url);
            if (mod && typeof mod.register === "function") {
                URL.revokeObjectURL(url);
                mod.register(tr);
                return "esm";
            }
        } finally {
            URL.revokeObjectURL(url);
        }
    } catch (_) { /* ignore and try classic */ }

    // 2) If classic path sees 'export', shim it into a classic wrapper
    let classicText = codeText;
    if (/^\s*export\s+function\s+register\s*\(\(/m.test(codeText)) {
        classicText =
            "window.trTerminalPackage={register:" +
            codeText
            .replace(/^\s*export\s+function\s+register\s*\(\(/m, "function(")
            .replace(/\}\s*$/, "}") + // keep body
            "}";
    } else if (/^\s*export\s+default\s+function\s*\(\(/m.test(codeText)) {
        classicText =
            "window.trTerminalPackage={register:" +
            codeText
            .replace(/^\s*export\s+default\s+function\s*\(\(/m, "function(")
            .replace(/\}\s*$/, "}") +
            "}";
    }

    // 3) Classic eval
    const fn = new Function("window", classicText + "\n;return\nwindow.trTerminalPackage;");
    const pkg = fn(window);
    if (pkg && typeof pkg.register === "function") {
        pkg.register(tr);
        try { delete window.trTerminalPackage; } catch { window.trTerminalPackage = undefined; }
        return "classic";
    }
    throw new Error("package has no register() export");
}

async function loadEnabledPackages() {
    // Load in CONFIG.order order
```

```
for (const name of CONFIG.order) {
    const meta = CONFIG.installed[name];
    if (!meta || !meta.enabled) continue;
    try {
        // Prefer packaged asset; otherwise localStorage blob
        let codeText = null;
        if (meta.filename) {
            try {
                const path = normalizePkgPath(meta.filename);
                codeText = await fetchSelf(path);
            } catch (_) { /* ignore */ }
        }
        if (!codeText && meta.local) {
            codeText = base64DecodeUtf8(meta.local);
        }
        if (!codeText) { print(`skip: ${name} (file not found)`, "muted");
continue; }
        const mode = await evalPackageText(name, codeText);
        print(`loaded: ${name} (${mode})`, "ok");
    } catch (e) {
        print(`load failed: ${name} - ${String(e.message || e)}`, "err");
    }
}
}

// ----- Panel: list + actions -----
function renderPkgList() {
    pkgList.innerHTML = "";
    if (CONFIG.order.length === 0) {
        const d = doc.createElement("div"); d.className = "muted"; d.textContent = "no packages installed";
        pkgList.appendChild(d); return;
    }
    CONFIG.order.forEach(name => {
        const meta = CONFIG.installed[name];
        if (!meta) return;
        const row = doc.createElement("div"); row.className = "row";
        row.style.padding = "6px 8px";
        const tag = doc.createElement("span"); tag.className = "tag";
        tag.textContent = `${meta.enabled ? "EN" : "DIS"} • ${name}`;
        const fn = doc.createElement("span"); fn.className = "muted"; fn.textContent = `${meta.filename || "(local only)"} `;
        const eb = doc.createElement("button"); eb.className = "btn"; eb.textContent = meta.enabled ? "Disable" : "Enable";
        const rb = doc.createElement("button"); rb.className = "btn"; rb.textContent = "Uninstall";
        eb.onclick = () => { if (!requireAdmin()) return; meta.enabled = !meta.enabled; meta.updatedAt = new Date().toISOString(); saveConfigLocal(); renderPkgList(); refreshBadges(); };
        rb.onclick = () => { if (!requireAdmin()) return; delete CONFIG.installed[name]; CONFIG.order = CONFIG.order.filter(n => n !== name); saveConfigLocal(); renderPkgList(); refreshBadges(); };
        row.append(tag, fn, eb, rb);
        pkgList.appendChild(row);
    })
}
```

```
        });
    }

    function renderConfigView() {
        configView.innerHTML = "";
        const pre = doc.createElement("pre");
        pre.textContent = JSON.stringify(CONFIG, null, 2);
        configView.appendChild(pre);
    }

    // ----- File pickers -----
    function pickFile(accept, cb) {
        const inp = doc.createElement("input");
        inp.type = "file"; inp.accept = accept;
        inp.onchange = async () => { const f = inp.files && inp.files[0]; if (!f) return; const txt = await f.text(); cb(f, txt); };
        inp.click();
    }

    // ----- Topbar buttons -----
    on($("#panelBtn"), "click", () => panel.classList.toggle("open"));
    on($("#adminBtn"), "click", () => { CTRL.admin = !CTRL.admin;
$($("#adminBtn").textContent = `Admin: ${CTRL.admin ? "on" : "off"} `;
refreshBadges()); });

    // Panel actions
    on($("#installBtn"), "click", () => {
        if (!requireAdmin()) return;
        pickFile(".js", async (file, text) => {
            const nameHint = ($("#fileName").value || file.name || "package.js").replace(/\s+/g, "-");
            const base = nameHint.replace(/\.js$/i, "");
            const name = uniqueName(base);
            const now = new Date().toISOString();
            const packagedFilename = normalizePkgPath(file.name || (base + ".js"));
            CONFIG.installed[name] = {
                filename: packagedFilename,
                enabled: true,
                type: "ext",
                hash: simpleHash(text),
                installedAt: now,
                updatedAt: now,
                local: base64EncodeUtf8(text)
            };
            if (!CONFIG.order.includes(name)) CONFIG.order.push(name);
            saveConfigLocal();
            renderPkgList(); refreshBadges();
            print(`installed: ${name} • ${packagedFilename} (place file under /extend/
when packaging)`, "ok");
            // Offer export so the file can be placed next to the app on disk
            download((file.name || (base + ".js")), text, "text/javascript");
        });
    });
}
```

```
function clearRegistry() {
    registry.commands = Object.create(null);
    registry.aliases = Object.create(null);
    // re-register built-ins
    registerCommand("echo", (argv, api) => api.print(argv.join(" ")));
    registerCommand("help", (argv, api) => api.print(coreHelp));
    registerCommand("state", (argv, api) => api.print(JSON.stringify({ admin:
CTRL.admin, version: "1.0.0" }, null, 2)));
    registerCommand("version", (argv, api) => api.print("TR Terminal v1.0.0"));
}

on($("#reloadBtn"), "click", async () => {
    clearRegistry(); // <- add this line
    await loadEnabledPackages();
    print("reload complete", "muted");
});

on($("#exportCfg"), "click", () => {
    download("tr-terminal-config.json", JSON.stringify(CONFIG, null, 2));
});

function normalizeConfigPaths() {
    for (const name of Object.keys(CONFIG.installed)) {
        const meta = CONFIG.installed[name];
        if (meta && meta.filename) meta.filename = normalizePkgPath(meta.filename);
    }
}

on($("#importCfg"), "click", () => {
    if (!requireAdmin()) return;
    pickFile("application/json,.json", async (file, text) => {
        try {
            const obj = JSON.parse(text);
            mergeConfig(obj);
            normalizeConfigPaths(); // <- add this
            saveConfigLocal();
            renderPkgList(); renderConfigView(); refreshBadges();
            print(`config merged: ${file.name}`, "ok");
        } catch (e) {
            print("config import failed: " + String(e.message || e), "err");
        }
    });
});

on($("#resetCfg"), "click", () => {
    if (!requireAdmin()) return;
    CONFIG = structuredClone(DEFAULT_CFG);
    saveConfigLocal(); renderPkgList(); renderConfigView(); refreshBadges();
    print("local config reset", "muted");
});

// ----- Utilities -----
// --- UTF-8 safe Base64 helpers ---
function base64EncodeUtf8(str) {
    if (window.TextEncoder) {
```

```
const bytes = new TextEncoder().encode(str);
let bin = "";
for (let i = 0; i < bytes.length; i++) bin += String.fromCharCode(bytes[i]);
return btoa(bin);
}

// Fallback (older engines)
return btoa(unescape(encodeURIComponent(str)));
}

function base64DecodeUtf8(b64) {
if (!b64) return "";
if (window.TextDecoder) {
const bin = atob(b64);
const bytes = new Uint8Array(bin.length);
for (let i = 0; i < bin.length; i++) bytes[i] = bin.charCodeAt(i);
return new TextDecoder().decode(bytes);
}
// Fallback (older engines)
return decodeURIComponent(escape(atob(b64)));
}

function uniqueName(base) {
let n = base.replace(/[^a-z0-9\-\_\.]/ig, "");
if (!n) n = "pkg";
let k = n, i = 1;
while (CONFIG.installed[k]) { k = `${n}-${i++}`; }
return k;
}

function simpleHash(text) {
let h = 2166136261 >>> 0;
for (let i = 0; i < text.length; i++) {
h ^= text.charCodeAt(i);
h = Math.imul(h, 16777619) >>> 0;
}
return ("00000000" + h.toString(16)).slice(-8);
}

// ----- REPL -----
const helpText = [
"OVERVIEW",
" Minimal TR Terminal with admin developer mode and package system.",
" Packages live next to tr-terminal.js (preferred) or in local storage
(dev).",
"",
"REPL COMMANDS",
" :help",
" :admin on|off           Toggle admin mode",
" :panel                  Toggle Packages panel",
" :list                   List installed packages",
" :install                Pick a .js package file (admin)",
" :enable <name>          Enable package (admin)",
" :disable <name>          Disable package (admin)",
" :uninstall <name>        Remove package (admin)",
" :reload                 Reload all enabled packages",
]
```

```
"  :config export|import|reset Config ops (import/reset are admin)",
"  :echo <...>                                Built-in test command",
"  :state                                     Show terminal state",
"  :version                                    Show version",
"",
"Packages may add more commands (run :help afterwards)."
].join("\n");

function runLine(line) {
const raw = String(line || "").trim();
if (!raw) return;
if (!raw.startsWith(":")) {
  print("(data) " + raw, "muted");
  return;
}
const parts = raw.slice(1).split(/\s+/);
const cmd = parts.shift().toLowerCase();
const args = parts;

// First try built-ins
switch (cmd) {
  case "help": return print(helpText);
  case "panel": panel.classList.toggle("open"); return;
  case "admin": {
    const onOff = (args[0] || "").toLowerCase();
    CTRL.admin = onOff === "on";
    $("#adminBtn").textContent = `Admin: ${CTRL.admin ? "on" : "off"}`;
    refreshBadges();
    return print(`admin=${onOff} ${CTRL.admin ? "on" : "off"}, ok`);
  }
  case "list": {
    const items = CONFIG.order.map(n => {
      const m = CONFIG.installed[n];
      return ` - ${n} [${m.enabled ? "enabled" : "disabled"}] ${m.filename || "(local)"}`;
    }).join("\n") || "(none)";
    return print(items);
  }
  case "install": {
    if (!requireAdmin()) return;
    $("#panelBtn").click(); // open panel to install
    return;
  }
  case "enable": {
    if (!requireAdmin()) return;
    const name = args[0]; if (!name) return print("usage: :enable <name>", "muted");
    const m = CONFIG.installed[name]; if (!m) return print("not found: " + name, "err");
    m.enabled = true; m.updatedAt = new Date().toISOString();
    saveConfigLocal(); renderPkgList(); refreshBadges();
    return print(`enabled: ${name}, ok`);
  }
  case "disable": {
```

```
    if (!requireAdmin()) return;
    const name = args[0]; if (!name) return print("usage: :disable <name>",
"muted");
    const m = CONFIG.installed[name]; if (!m) return print("not found: " +
name, "err");
    m.enabled = false; m.updatedAt = new Date().toISOString();
saveConfigLocal(); renderPkgList(); refreshBadges();
    return print("disabled: " + name, "ok");
}
case "uninstall": {
    if (!requireAdmin()) return;
    const name = args[0]; if (!name) return print("usage: :uninstall <name>",
"muted");
    if (!CONFIG.installed[name]) return print("not found: " + name, "err");
    delete CONFIG.installed[name];
    CONFIG.order = CONFIG.order.filter(n => n !== name);
    saveConfigLocal(); renderPkgList(); refreshBadges();
    return print("uninstalled: " + name, "ok");
}
case "reload": return loadEnabledPackages().then(() => print("reload
complete", "muted"));
case "config": {
    const sub = (args[0] || "").toLowerCase();
    if (sub === "export") { download("tr-terminal-config.json",
JSON.stringify(CONFIG, null, 2)); return; }
    if (sub === "import") {
        if (!requireAdmin()) return;
        pickFile("application/json,.json", async (file, text) => {
            try { const obj = JSON.parse(text); mergeConfig(obj);
saveConfigLocal(); renderPkgList(); renderConfigView(); refreshBadges();
print(`config merged: ${file.name}`, "ok"); }
            catch (e) { print("config import failed: " + String(e.message || e),
"err"); }
        });
        return;
    }
    if (sub === "reset") {
        if (!requireAdmin()) return;
        CONFIG = structuredClone(DEFAULT_CFG);
        saveConfigLocal(); renderPkgList(); renderConfigView(); refreshBadges();
        return print("local config reset", "muted");
    }
    return print("usage: :config export|import|reset", "muted");
}
default:
    // Aliases?
    if (registry.aliases[cmd]) {
        const expanded = registry.aliases[cmd] + (args.length ? (" " +
args.join(" ")) : "");
        return runLine(": " + expanded);
    }
    // Packages?
    const fn = registry.commands[cmd];
    if (typeof fn === "function") {
```

```

        try { return fn(args, apiObject()); }
        catch (e) { return print("cmd error: " + String(e.message || e), "err");
    }
}
return print("unknown command: :" + cmd + " (try :help)", "err");
}

// -----
print("TR Terminal ready. Type :help");
refreshBadges();
renderPkgList();
renderConfigView();
loadEnabledPackages();

// -----
doc.addEventListener("keydown", (e) => {
    if ((e.ctrlKey || e.metaKey) && e.key.toLowerCase() === "p") {
        e.preventDefault(); panel.classList.toggle("open");
    }
});
on(cmdEl, "keydown", (e) => {
    if (e.key === "Enter") {
        e.preventDefault();
        const v = cmdEl.value; cmdEl.value = "";
        print("> " + v, "muted");
        runLine(v);
    }
});
}
}

```

## [58] GenesisOS/tr-terminal.json

- **Bytes:** 401
- **Type:** text

```
{
    "id": "tr-terminal",
    "title": "TR Terminal",
    "description": "Scalable Genesis OS REPL with patch and package extension system.",
    "version": "1.0.0",
    "author": "You",
    "icon": "\ud83d\udda5\ufe0f",
    "category": "Development",
    "entry": "tr-terminal.js",
    "permissions": [],
    "window": {
        "width": 900,
        "height": 600,
    }
}
```

```
        "resizable": true
    }
}
```

## [59] GenesisOS/tr-VersionManager.php

- **Bytes:** 8373
- **Type:** text

```
<?php
class VersionManager {
    private string $livePath;
    private string $versionsPath;

    public function __construct(string $livePath, string $versionsPath) {
        if (!is_dir($livePath)) throw new Exception("Live BEP path does not exist: $livePath");
        if (!is_writable($versionsPath)) throw new Exception("Versions directory is not writable: $versionsPath");

        $this->livePath = rtrim($livePath, '/');
        $this->versionsPath = rtrim($versionsPath, '/');
    }

    private function getVersionPath(string $version): string {
        if (empty($version) || str_contains($version, '..') || str_contains($version, '/')) {
            throw new Exception("Invalid version name provided.");
        }
        $path = $this->versionsPath . '/' . $version;
        if (!is_dir($path)) {
            throw new Exception("Version '$version' not found.");
        }
        return $path;
    }

    public function listVersions(): array {
        $dirs = array_filter(scandir($this->versionsPath), fn($d) => $d[0] !== '.' && is_dir($this->versionsPath . '/' . $d));
        rsort($dirs); // Show newest first
        return array_values($dirs);
    }

    //  FIXED: Changed 'string' to '?string' to explicitly declare the parameter as nullable.
    public function createVersion(?string $name = null): string {
        $versionName = $name ?: date('Ymd_His');
        $targetPath = $this->versionsPath . '/' . $versionName;
        if (is_dir($targetPath)) throw new Exception("Version '$versionName' already exists.");
    }
}
```

```
$this->recursiveCopy($this->livePath, $targetPath);
return $versionName;
}

public function listVersionFiles(string $version): array {
$versionPath = $this->getVersionPath($version);
$files = [];
$iterator = new RecursiveIteratorIterator(
    new RecursiveDirectoryIterator($versionPath,
RecursiveDirectoryIterator::SKIP_DOTS),
    RecursiveIteratorIterator::SELF_FIRST
);

foreach ($iterator as $item) {
    $files[] = [
        'path' => $iterator->getSubPathName(),
        'type' => $item->isDir() ? 'dir' : 'file'
    ];
}
return $files;
}

public function readFile(string $version, string $filePath): string {
$versionPath = $this->getVersionPath($version);
$fullPath = $versionPath . '/' . $filePath;

if (str_contains($filePath, '..') || !file_exists($fullPath) ||
is_dir($fullPath)) {
    throw new Exception("File not found or invalid path.");
}
return file_get_contents($fullPath);
}

public function writeFile(string $version, string $filePath, string $content): void {
$versionPath = $this->getVersionPath($version);
$fullPath = $versionPath . '/' . $filePath;

if (str_contains($filePath, '..')) {
    throw new Exception("Invalid file path.");
}

$dir = dirname($fullPath);
if (!is_dir($dir)) {
    mkdir($dir, 0755, true);
}

if (file_put_contents($fullPath, $content) === false) {
    throw new Exception("Failed to write to file.");
}
}

public function runTests(string $version): array {
$versionPath = $this->getVersionPath($version);
```

```
$results = [];

// Test 1: Check for PHP syntax errors in all .php files
$php_files_ok = true;
$syntax_errors = [];
$iterator = new RecursiveIteratorIterator(new
RecursiveDirectoryIterator($versionPath));
foreach ($iterator as $file) {
    if ($file->getExtension() === 'php') {
        $output = [];
        exec('php -l ' . escapeshellarg($file->getPathname()), $output,
$return_var);
        if ($return_var !== 0) {
            $php_files_ok = false;
            $syntax_errors[] = implode("\n", $output);
        }
    }
}
$results['PHP Syntax Check'] = ['passed' => $php_files_ok, 'details' =>
$php_files_ok ? 'All files OK.' : implode("\n", $syntax_errors)];

// Test 2: Essential file existence
$essential_files = ['index.php', 'config.php', 'bootstrap.php',
'modules/auth.php', 'gos_files/system'];
$missing_files = [];
foreach ($essential_files as $file) {
    if (!file_exists($versionPath . '/' . $file)) {
        $missing_files[] = $file;
    }
}
$results['Essential Files Check'] = ['passed' => empty($missing_files),
'details' => empty($missing_files) ? 'All present.' : 'Missing: ' . implode(', ', $missing_files)];

return $results;
}

public function deployVersion(string $version): void {
    $sourcePath = $this->getVersionPath($version);

    // 1. Create a backup before deploying
    $this->createVersion('backup_before_deploy_' . $version . '_' .
date('His'));

    // 2. Delete current live files
    $this->recursiveDelete($this->livePath);

    // 3. Copy new version to live
    $this->recursiveCopy($sourcePath, $this->livePath);
}

public function deleteVersion(string $version): void {
    $path = $this->getVersionPath($version);
    $this->recursiveDelete($path, true);
```

```
}

public function downloadVersion(string $version): void {
    $versionPath = $this->getVersionPath($version);

    $zipFile = sys_get_temp_dir() . '/' . $version . '.zip';
    if (file_exists($zipFile)) {
        unlink($zipFile);
    }

    $zip = new ZipArchive();
    $zip->open($zipFile, ZipArchive::CREATE | ZipArchive::OVERWRITE);

    $files = new RecursiveIteratorIterator(
        new RecursiveDirectoryIterator($versionPath),
        RecursiveIteratorIterator::LEAVES_ONLY
    );

    foreach ($files as $name => $file) {
        if (!$file->isDir()) {
            $filePath = $file->getRealPath();
            $relativePath = substr($filePath, strlen($versionPath) + 1);
            $zip->addFile($filePath, $relativePath);
        }
    }
    $zip->close();

    header('Content-Type: application/zip');
    header('Content-Disposition: attachment; filename=' . $version .
'.zip');
    header('Content-Length: ' . filesize($zipFile));
    readfile($zipFile);
    unlink($zipFile);
    exit;
}

private function recursiveCopy(string $source, string $dest): void {
    if (!is_dir($dest)) mkdir($dest, 0755, true);

    $iterator = new RecursiveIteratorIterator(
        new RecursiveDirectoryIterator($source,
RecursiveDirectoryIterator::SKIP_DOTS),
        RecursiveIteratorIterator::SELF_FIRST
    );

    foreach ($iterator as $item) {
        $destPath = $dest . '/' . $iterator->getSubPathName();
        if ($item->isDir()) {
            if (!is_dir($destPath)) mkdir($destPath);
        } else {
            copy($item, $destPath);
        }
    }
}
```

```

    private function recursiveDelete(string $dir, bool $deleteSelf = false): void
{
    if (!is_dir($dir)) return;

    $files = new RecursiveIteratorIterator(
        new RecursiveDirectoryIterator($dir,
            RecursiveDirectoryIterator::SKIP_DOTS),
        RecursiveIteratorIterator::CHILD_FIRST
    );

    foreach ($files as $fileinfo) {
        $todo = ($fileinfo->isDir() ? 'rmdir' : 'unlink');
        $todo($fileinfo->getRealPath());
    }

    if ($deleteSelf) {
        rmdir($dir);
    }
}
}

// Special handler for download requests, which are not AJAX calls.
if (isset($_GET['action']) && $_GET['action'] === 'download' &&
isset($_GET['version'])) {
    require_once 'auth.php';
    if (isLoggedIn()) {
        $live_tr_path = realpath(__DIR__ . '/../tr');
        $versions_dir = __DIR__ . '/versions';
        $manager = new VersionManager($live_tr_path, $versions_dir);
        $manager->downloadVersion($_GET['version']);
    }
}
}

```

## [60] GenesisOS/users.php

- **Bytes:** 8020
- **Type:** text

```

<?php
/**
 * Text Reversion
 * Users Module
 */

/**
 * Handle users API requests.
 * This provides endpoints for administrators to manage user accounts.
 */
function handleUsersAPI() {
    $method = $_POST['method'] ?? '';

```

```
$params = $_POST['params'] ?? [];

if (is_string($params)) {
    $params = json_decode($params, true) ?? [];
}

// Most methods require admin privileges.
if (!isAdmin()) {
    // Exception: A user can change their own password.
    if ($method === 'setPassword') {
        $user = getCurrentUser();
        // Allow if the target user is the current user.
        if (isset($params['username']) && $user['username'] ===
$params['username']) {
            return setPassword($params);
        }
    }
    return ['success' => false, 'message' => 'Access denied'];
}

// Admin-only functions
switch ($method) {
    case 'listUsers':
        $users = loadUserDB();
        // Never expose passwords via the API.
        foreach ($users as &$u) {
            unset($u['password']);
        }
        return ['success' => true, 'data' => $users];

    case 'createUser':
        return createUser($params);

    //  NEW: Added case for getting a single user's details.
    case 'getUserDetails':
        return getUserDetails($params);

    case 'updateUser':
        return updateUser($params);

    case 'deleteUser':
        return deleteUser($params);

    case 'setPassword':
        return setPassword($params);

    default:
        return ['success' => false, 'message' => 'Invalid user management
method.'];
}
}

/** 
 *  NEW: Retrieves a single user's details. (Admin only)

```

```
/*
function getUserDetails(array $params): array {
    $username = $params['username'] ?? null;
    if (!$username) {
        return ['success' => false, 'message' => 'Username is required.'];
    }

    $users = loadUserDB();
    foreach ($users as $user) {
        if ($user['username'] === $username) {
            unset($user['password']); // Never expose the password hash.
            return ['success' => true, 'data' => $user];
        }
    }

    return ['success' => false, 'message' => 'User not found.'];
}

/** 
 * Creates a new user. (Admin only)
 */
function createUser(array $params): array {
    $username = $params['username'] ?? null;
    $password = $params['password'] ?? null;
    $name = $params['name'] ?? 'New User';
    $roles = $params['roles'] ?? ['user'];

    if (!$username || !$password) {
        return ['success' => false, 'message' => 'Username and password are required.'];
    }
    if (!is_array($roles)) {
        return ['success' => false, 'message' => 'Roles must be an array.'];
    }

    $users = loadUserDB();
    foreach ($users as $user) {
        if ($user['username'] === $username) {
            return ['success' => false, 'message' => 'Username already exists.'];
        }
    }

    $users[] = [
        'username' => $username,
        'password' => password_hash($password, PASSWORD_DEFAULT),
        'roles' => $roles,
        'name' => htmlspecialchars($name),
        'quota' => 10 * 1024 * 1024, // Default 10MB
        'lastLogin' => null
    ];
    saveUserDB($users);

    // Also create their home directory
}
```

```
global $config;
$userDir = $config['filesystem']['user_dir'] . '/' . $username;
if (!is_dir($userDir)) {
    mkdir($userDir, 0755, true);
    mkdir($userDir . '/Desktop', 0755, true);
    mkdir($userDir . '/Documents', 0755, true);
}

return ['success' => true, 'message' => 'User created successfully.'];
}

/***
 * Updates an existing user's details. (Admin only)
 */
function updateUser(array $params): array {
    $username = $params['username'] ?? null;
    if (!$username) {
        return ['success' => false, 'message' => 'Username is required.'];
    }

    $users = loadUserDB();
    $found = false;
    foreach ($users as $i => &$user) {
        if ($user['username'] === $username) {
            $found = true;
            // Update name if provided
            if (isset($params['name'])) {
                $user['name'] = htmlspecialchars($params['name']);
            }
            // Update quota if provided
            if (isset($params['quota'])) {
                $user['quota'] = intval($params['quota']);
            }
            // Update roles if provided, with protection for the admin user
            if (isset($params['roles']) && is_array($params['roles'])) {
                if ($user['username'] === 'admin' && !in_array('admin',
$params['roles'])) {
                    return ['success' => false, 'message' => 'Cannot remove admin
role from the primary administrator.'];
                }
                $user['roles'] = $params['roles'];
            }
            break;
        }
    }

    if ($found) {
        saveUserDB($users);
        return ['success' => true, 'message' => 'User updated successfully.'];
    }

    return ['success' => false, 'message' => 'User not found.'];
}
```

```
/*
 * Deletes a user. (Admin only)
 */
function deleteUser(array $params): array {
    $username = $params['username'] ?? null;
    if (!$username) {
        return ['success' => false, 'message' => 'Username is required.'];
    }
    if ($username === 'admin') {
        return ['success' => false, 'message' => 'Cannot delete the primary
administrator account.'];
    }

    $users = loadUserDB();
    $initialCount = count($users);
    $users = array_filter($users, fn($user) => $user['username'] !== $username);

    if (count($users) < $initialCount) {
        saveUserDB(array_values($users));
        // Optional: Recursively delete user's directory for complete removal.
        // Be very careful with this in a production environment.
        // global $config;
        // $userDir = $config['filesystem']['user_dir'] . '/' . $username;
        // if (is_dir($userDir)) { /* recursive delete logic here */ }
        return ['success' => true, 'message' => 'User deleted successfully.'];
    }

    return ['success' => false, 'message' => 'User not found.'];
}

/*
 * Sets a new password for a user.
 * Can be called by an admin for any user, or by a user for themselves.
 */
function setPassword(array $params): array {
    $username = $params['username'] ?? null;
    $newPassword = $params['newPassword'] ?? null;
    $currentPassword = $params['currentPassword'] ?? null;

    if (!$username || !$newPassword) {
        return ['success' => false, 'message' => 'Username and new password are
required.'];
    }
    if (strlen($newPassword) < 8) {
        return ['success' => false, 'message' => 'New password must be at least 8
characters.'];
    }

    $currentUser = getCurrentUser();
    $isSelfChange = $currentUser['username'] === $username;

    $users = loadUserDB();
    $found = false;
    foreach ($users as $i => &$user) {
```

```

        if ($user['username'] === $username) {
            $found = true;
            // If user is changing their own password, verify the current one.
            if ($isSelfChange && !isAdmin()) {
                if (!$currentPassword || !password_verify($currentPassword,
$user['password'])) {
                    return ['success' => false, 'message' => 'Incorrect current
password.'];
                }
            }
            // Admins can change passwords without the current one.
            $user['password'] = password_hash($newPassword, PASSWORD_DEFAULT);
            break;
        }
    }

    if ($found) {
        saveUserDB($users);
        return ['success' => true, 'message' => 'Password updated successfully.'];
    }

    return ['success' => false, 'message' => 'User not found.'];
}

```

## [61] GenesisOS/x000000000.txt

- **Bytes:** 4673
- **Type:** text

```

/**
 * @file main.c
 * @brief Permutation Generator in C, architected to safety-critical standards.
 *
 * @author Dominic Alexander Cooper (Original Algorithm)
 * @author Gemini (Architectural Refactoring)
 *
 * @details
 * This program generates all possible character combinations for a given length,
 * based on a predefined character set. It is a complete rewrite of the original
 * concept to align with principles of safety-critical software design.
 */
#include <stdio.h>
#include <stdint.h> // For fixed-width integer types like uint64_t
#include <stdbool.h> // For bool type
#include <limits.h> // For UINT64_MAX
//=====
// 1. CONFIGURATION AND DATA DEFINITIONS
//=====
#define MAX_PERMUTATION_LENGTH 10

```

```

static const char ALPHABET[] = {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
    'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
    'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '\n'
};

static const uint64_t ALPHABET_SIZE = sizeof(ALPHABET) / sizeof(ALPHABET[0]);
//=====

// 2. ABSTRACT INTERFACE FOR OUTPUT (OutputSink)
//=====

typedef struct {
    void* context;
    bool (*write)(void* context, const char* buffer, uint64_t size);
    bool (*write_char)(void* context, char c);
} OutputSink;
//=====

// 3. CORE LOGIC (Generator)
//=====

static bool safe_uint64_power(uint64_t base, uint64_t exp, uint64_t* result) {
    *result = 1;
    for (uint64_t i = 0; i < exp; ++i) {
        if (*result > UINT64_MAX / base) {
            return false;
        }
        *result *= base;
    }
    return true;
}

static bool generate_permutations(uint64_t length, OutputSink* sink) {
    uint64_t num_combinations;
    if (!safe_uint64_power(ALPHABET_SIZE, length, &num_combinations)) {
        return false;
    }
    char current_perm[MAX_PERMUTATION_LENGTH];
    for (uint64_t i = 0; i < num_combinations; ++i) {
        uint64_t temp_row = i;
        for (int64_t j = length - 1; j >= 0; --j) {
            uint64_t char_index = temp_row % ALPHABET_SIZE;
            current_perm[j] = ALPHABET[char_index];
            temp_row /= ALPHABET_SIZE;
        }
        if (!sink->write(sink->context, current_perm, length)) {
            return false;
        }
        if (!sink->write_char(sink->context, '\n')) {
            return false;
        }
    }
    return true;
}

//=====

// 4. CONCRETE IMPLEMENTATION OF OUTPUTSINK (FileSink)
//=====

```

```
static bool file_sink_write(void* context, const char* buffer, uint64_t size) {
    FILE* p = (FILE*)context;
    return fwrite(buffer, sizeof(char), size, p) == size;
}
static bool file_sink_write_char(void* context, char c) {
    FILE* p = (FILE*)context;
    return fputc(c, p) != EOF;
}
static bool file_sink_init(OutputSink* sink, const char* filename) {
    FILE* p = fopen(filename, "w");
    if (p == NULL) {
        return false;
    }
    *sink = (OutputSink){
        .context = p,
        .write = file_sink_write,
        .write_char = file_sink_write_char
    };
    return true;
}
static void file_sink_close(OutputSink* sink) {
    if (sink && sink->context) {
        fclose((FILE*)sink->context);
        sink->context = NULL;
    }
}
//=====================================================================
// 5. SYSTEM ASSEMBLER (main)
//=====================================================================
int main(void) {
    const uint64_t permutation_length = 4;
    if (permutation_length == 0 || permutation_length > MAX_PERMUTATION_LENGTH) {
        return 1;
    }
    OutputSink file_sink;
    if (!file_sink_init(&file_sink, "system_safe.txt")) {
        perror("Error opening file");
        return 1;
    }
    bool success = generate_permutations(permutation_length, &file_sink);
    file_sink_close(&file_sink);
    if (!success) {
        return 1;
    }
    return 0;
}
```

## [62] GenesisOS/x000000001.txt

- **Bytes:** 6642
- **Type:** text

```
//=====
// DSL ( 仕様メタ情報 )
//=====
/*
b000000000 (言葉){

    r000000000 [

        a000000000 青写真
        a000000001 天国
        a000000002 神
        a000000003 神の王国
        a000000004 変数と定数
        a000000005 ハッシュテーブル

    ]

}

*/
//=====

// ファイル名: main.cpp
// 概要: 安全性重視の組合せ (順列的直積) ジェネレータ (日本語版)
// 著者: Dominic Alexander Cooper (原案) / Gemini (構造再設計)
// 設計方針: 機能分離 (出力抽象・生成器・具象出力・組立) とオーバーフロー防止
//=====

#include <cstdint> // std::uint64_t
#include <cstdio> // std::FILE, std::fopen, std::fwrite, std::fputc,
std::fclose, std::perror
#include <limits> // std::numeric_limits
#include <array> // std::array
#include <cinttypes> // PRIu64

//=====

// 1. 構成・データ定義
//=====

// 最大長 (ビルド時定数)
constexpr std::size_t 最大列長 = 10;

// 文字集合 (改行と空白を含む)
static const char 文字集合[] = {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
    'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
    'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '\n'
};
static const std::uint64_t 文字集合数 =
    static_cast<std::uint64_t>(sizeof(文字集合) / sizeof(文字集合[0]));
```

```
//=====
// 2. 出力抽象（出力先）
//=====

struct 出力先 {
    void* 文脈; // 出力コンテキスト（例: FILE*)
    bool (*書く)(void* 文脈, const char* バッファ, std::uint64_t サイズ);
    bool (*一文字書く)(void* 文脈, char 文字);
};

//=====
// 3. 中核ロジック（生成器）
//=====

// 安全べき乗（オーバーフロー検出付き：結果 = 基数^指数）
static bool 安全べき乗(std::uint64_t 基数, std::uint64_t 指数, std::uint64_t* 結果)
{
    if (!結果) return false;
    *結果 = 1;
    for (std::uint64_t i = 0; i < 指数; ++i) {
        if (*結果 > (std::numeric_limits<std::uint64_t>::max)() / 基数) {
            return false; // オーバーフロー
        }
        *結果 *= 基数;
    }
    return true;
}

// 組合せ生成（長さ = 列長）。各列を出力し、末尾に改行を付加。
static bool 組合せを生成(std::uint64_t 列長, 出力先* 先) {
    if (!先 || !先->書く || !先->一文字書く) return false;

    // 総組合せ数 = |Σ|^列長（オーバーフロー検査付き）
    std::uint64_t 総数 = 0;
    if (!安全べき乗(文字集合数, 列長, &総数)) {
        return false;
    }

    // 作業バッファ（最大列長はビルド時制限）
    if (列長 == 0 || 列長 > 最大列長) return false;
    char 現在列[最大列長];

    for (std::uint64_t 通番 = 0; 通番 < 総数; ++通番) {
        std::uint64_t 作業 = 通番;

        // |Σ| 進数で通番を分解し、右詰めで文字列化
        for (std::int64_t 位置 = static_cast<std::int64_t>(列長) - 1; 位置 >= 0; --位置) {
            std::uint64_t 指標 = 作業 % 文字集合数;
            現在列[位置] = 文字集合[指標];
            作業 /= 文字集合数;
        }

        // 列を書き出し
        if (!先->書く(先->文脈, 現在列, 列長)) {

```

```
        return false;
    }
    // 行区切り(改行)
    if (!先->一文字書く(先->文脈, '\n')) {
        return false;
    }
}
return true;
}

//=====
// 4. 出力先の具象実装(ファイル出力)
//=====

static bool ファイルへ書く(void* 文脈, const char* バッファ, std::uint64_t サイズ) {
    std::FILE* fp = static_cast<std::FILE*>(文脈);
    if (!fp) return false;
    return std::fwrite(バッファ, sizeof(char), static_cast<std::size_t>(サイズ), fp)
        == static_cast<std::size_t>(サイズ);
}

static bool ファイルへ一文字(void* 文脈, char 文字) {
    std::FILE* fp = static_cast<std::FILE*>(文脈);
    if (!fp) return false;
    return std::fputc(static_cast<unsigned char>(文字), fp) != EOF;
}

static bool ファイル出力先を初期化(出力先* 先, const char* ファイル名) {
    if (!先 || !ファイル名) return false;
    std::FILE* fp = std::fopen(ファイル名, "w");
    if (!fp) return false;
    *先 = 出力先{
        /*文脈*/         fp,
        /*書く*/         &ファイルへ書く,
        /*一文字書く*/   &ファイルへ一文字
    };
    return true;
}

static void ファイル出力先を閉じる(出力先* 先) {
    if (先 && 先->文脈) {
        std::fclose(static_cast<std::FILE*>(先->文脈));
        先->文脈 = nullptr;
    }
}

//=====
// 5. システム組立(main)
//=====

int main() {
    // 生成する列長(必要に応じて変更)
    constexpr std::uint64_t 列長 = 4;

    if (列長 == 0 || 列長 > 最大列長) {
        return 1; // 不正な長さ
    }
}
```

```

    }

    出力先 ファイル先{};
    if (!ファイル出力先を初期化(&ファイル先, "system_safe.txt")) {
        std::cerr("ファイルを開けません");
        return 1;
    }

    const bool 成功 = 組合せを生成(列長, &ファイル先);
    ファイル出力先を閉じる(&ファイル先);

    if (!成功) {
        return 1;
    }
    return 0;
}

```

## [63] Hardware/C700h-stable.py

- **Bytes:** 19311
- **Type:** text

```

#!/usr/bin/env python3
"""

C700h.py (Python 3.14.2)

C700 Host utility: Acceptance → Deterministic Quantum/Classical Circuit
Derivation.

Rewritten from the uploaded dimension-generator.py:
- Same acceptance rules (7-digit tokenization, perfect-square grid, color range,
adjacency conflict).
- Same deterministic mapping into Qiskit circuits and optional classical "shadow".
- Adds a clean CLI with subcommands: accept, derive, menu.
- Still supports huge integer accepted-states (decimal) and binary "0b...".

Dependencies (for derive PNG output):
    pip install "qiskit[visualization]" matplotlib numpy pillow

Notes:
- Qiskit is imported lazily so `accept` works without Qiskit installed.
- Output defaults to ./out
"""

from __future__ import annotations

import argparse
import json
import math
import os

```

```
import re
import sys
from dataclasses import dataclass
from typing import Any, Iterable, List, Optional, Sequence, Tuple

# -----
# Constants
# -----

SEGMENT_LEN: int = 7
MAX_COLOR_INDEX: int = 16 ** 6 # 16777216

GATES: List[str] = ["x", "y", "z", "h", "s", "sdg", "t", "tdg", "rx", "ry", "rz",
"cx"]

# -----
# Acceptance helpers
# -----

def is_perfect_square(n: int) -> bool:
    if n <= 0:
        return False
    r = math.sqrt(n)
    return r * r == n

def decode_id_to_color_indexes(id_string: str, segment_length: int = SEGMENT_LEN)
-> List[int]:
    """
    Mirrors map.js behavior:
    - Split left-to-right into fixed segment_length chunks.
    - No padding.
    - Last segment may be shorter.
    - Non-integer segments are skipped.
    """
    out: List[int] = []
    for i in range(0, len(id_string), segment_length):
        seg = id_string[i : i + segment_length]
        try:
            out.append(int(seg, 10))
        except ValueError:
            # keep behavior: silently ignore invalid chunks
            pass
    return out

def are_valid_color_indexes(indexes: Sequence[int]) -> bool:
    return all(1 <= x <= MAX_COLOR_INDEX for x in indexes)

def has_adjacent_conflict(indexes: Sequence[int], wrap: bool = False) -> bool:
    """
```

```
True if:
- token count not perfect square, OR
- any orthogonal neighbor pair in the grid is equal.
Optional wrap-around adjacency on right and bottom edges.
"""

if not is_perfect_square(len(indexes)):
    return True

m = math.sqrt(len(indexes))

def idx(r: int, c: int) -> int:
    return indexes[r * m + c]

for r in range(m):
    for c in range(m):
        cur = idx(r, c)

        # right
        if c + 1 < m:
            if cur == idx(r, c + 1):
                return True
        elif wrap and m > 1:
            if cur == idx(r, 0):
                return True

        # down
        if r + 1 < m:
            if cur == idx(r + 1, c):
                return True
        elif wrap and m > 1:
            if cur == idx(0, c):
                return True

return False

def color_index_to_hex(index: int) -> str:
    # 1 -> #000000, 16^6 -> #FFFFFF
    v = index - 1
    return "#" + format(v, "06x")

@dataclass(frozen=True)
class AcceptanceReport:
    ok: bool
    reason: str
    m: int
    indexes: List[int]
    hex_colors: List[str]

    def to_dict(self) -> dict[str, Any]:
        return {
            "ok": self.ok,
            "reason": self.reason,
```

```
        "m": self.m,
        "token_count": len(self.indexes),
        "indexes": self.indexes,
        "hex_colors": self.hex_colors,
    }

def verify_acceptance_from_decimal_string(id_decimal: str, wrap_adjacency: bool = False) -> AcceptanceReport:
    indexes = decode_id_to_color_indexes(id_decimal, segment_length=SEGMENT_LEN)

    if not is_perfect_square(len(indexes)):
        return AcceptanceReport(
            False,
            f"Token count {len(indexes)} is not a perfect square.",
            0,
            list(indexes),
            [],
        )

    m = math.sqrt(len(indexes))

    if not are_valid_color_indexes(indexes):
        return AcceptanceReport(
            False,
            "One or more tokens are outside [1..16^6].",
            m,
            list(indexes),
            [],
        )

    if has_adjacent_conflict(indexes, wrap=wrap_adjacency):
        return AcceptanceReport(
            False,
            "Adjacency conflict: at least one orthogonal neighbor pair is equal.",
            m,
            list(indexes),
            [],
        )

    hex_colors = [color_index_to_hex(x) for x in indexes]
    return AcceptanceReport(True, "Accepted.", m, list(indexes), hex_colors)

# -----
# Deterministic circuit derivation
# -----


def token_to_angle(token: int) -> float:
    # Deterministic discrete angle: multiples of pi/16 (never 0)
    k = (token % 32) + 1
    return k * (math.pi / 16.0)
```

```
def derive_quantum_and_classical_from_grid(
    indexes: Sequence[int],
    m: int,
    max_qubits: int = 8,
    max_layers: int = 16,
    reversible_only: bool = False,
):
    """
    Deterministic mapping:
    - Qubits = min(m, max_qubits)
    - Layers = min(m, max_layers)
    - Gate for (r,c) chosen by token % len(GATES)

    classical_shadow keeps:
    - x and cx when they occur (and in reversible_only mode)
    """

    # Lazy import so accept/help can run without qiskit installed
    from qiskit import QuantumCircuit # type: ignore

    q = min(m, max_qubits)
    layers = min(m, max_layers)

    qc = QuantumCircuit(q, name="derived_quantum")
    cc = QuantumCircuit(q, name="classical_shadow")

    def grid_token(r: int, c: int) -> int:
        return indexes[r * m + c]

    for r in range(layers):
        for c in range(q):
            tok = grid_token(r, c)
            gate = GATES[tok % len(GATES)]

            if reversible_only:
                # force into exact classical correspondence
                gate = "cx" if (tok % 2 == 1 and q > 1) else "x"

            if gate in ("x", "y", "z", "h", "s", "sdg", "t", "tdg"):
                getattr(qc, gate)(c)
                if gate == "x":
                    cc.x(c)

            elif gate == "cx":
                if q == 1:
                    qc.x(c)
                    cc.x(c)
                else:
                    control = c
                    shift = 1 + (tok % (q - 1))
                    target = (c + shift) % q
                    qc.cx(control, target)
                    cc.cx(control, target)

            elif gate in ("rx", "ry", "rz"):
```

```
        ang = token_to_angle(tok)
        getattr(qc, gate)(ang, c)
        # no classical equivalent in shadow

    qc.barrier()

    return qc, cc

# -----
# Output helpers
# -----


def ensure_dir(path: str) -> None:
    os.makedirs(path, exist_ok=True)


def save_circuit_png(qc, path: str) -> None:
    from qiskit.visualization import circuit_drawer # type: ignore
    circuit_drawer(qc, output="mpl", filename=path)


def combine_pngs_side_by_side(left_path: str, right_path: str, out_path: str) ->
None:
    from PIL import Image # type: ignore

    a = Image.open(left_path).convert("RGBA")
    b = Image.open(right_path).convert("RGBA")

    w = a.width + b.width
    h = max(a.height, b.height)
    out = Image.new("RGBA", (w, h), (20, 20, 20, 255))
    out.paste(a, (0, 0))
    out.paste(b, (a.width, 0))
    out.save(out_path)


def write_gate_sequence(qc, out_txt: str) -> None:
    """
    Qiskit 1.x compatible:
    - qc.data yields CircuitInstruction objects
    - Qubit index obtained via qc.find_bit(qubit).index
    """
    parts: List[str] = []

    for ci in qc.data:
        op = ci.operation
        if op.name == "barrier":
            continue

        qinds = [qc.find_bit(q).index for q in ci.qubits]

        if op.name in ("rx", "ry", "rz"):
            angle = op.params[0]
```

```
        parts.append(f"{op.name}({angle},{qinds[0]})")
    elif op.name == "cx":
        parts.append(f"cx({qinds[0]},{qinds[1]})")
    else:
        parts.append(f"{op.name}({qinds[0]})")

    with open(out_txt, "w", encoding="utf-8") as f:
        f.write(" ".join(parts) + "\n")

# -----
# Input parsing
# -----


_DEC_RE = re.compile(r"^[0-9]+$")

def parse_accepted_state_to_decimal_string(raw: str) -> str:
    """
    Accepts:
    - decimal integer string (possibly huge)
    - binary with 0b prefix
    Returns canonical decimal string (no leading zeros except "0").
    """
    s = raw.strip()
    if s.lower().startswith("0b"):
        n = int(s, 2)
        return str(n)

    if not _DEC_RE.fullmatch(s):
        raise ValueError("Not a valid decimal integer string (or 0b... binary).")

    # allow huge integers; normalize leading zeros
    s2 = s.lstrip("0")
    return s2 if s2 != "" else "0"

def read_text_file(path: str) -> str:
    with open(path, "r", encoding="utf-8") as f:
        return f.read().strip()

# -----
# Commands
# -----


def cmd_accept(args: argparse.Namespace) -> int:
    raw = read_text_file(args.infile) if args.infile else args.state
    if raw is None:
        print("No state provided. Use --state or --in.", file=sys.stderr)
        return 2

    try:
        dec_str = parse_accepted_state_to_decimal_string(raw)
```

```
except ValueError as e:
    print(str(e), file=sys.stderr)
    return 2

report = verify_acceptance_from_decimal_string(dec_str,
wrap_adjacency=args.wrap)
if args.json:
    print(json.dumps(report.to_dict(), indent=2))
else:
    print(report.reason)
    if report.ok:
        print(f"Grid: {report.m} x {report.m} (tokens={len(report.indexes)})")
    if args.show_hex:
        print("Hex colors:")
        print(" ".join(report.hex_colors))
    if args.show_indexes:
        print("Indexes:")
        print(" ".join(str(x) for x in report.indexes))

return 0 if report.ok else 1

def cmd_derive(args: argparse.Namespace) -> int:
    raw = read_text_file(args.infile) if args.infile else args.state
    if raw is None:
        print("No state provided. Use --state or --in.", file=sys.stderr)
        return 2

    try:
        dec_str = parse_accepted_state_to_decimal_string(raw)
    except ValueError as e:
        print(str(e), file=sys.stderr)
        return 2

    report = verify_acceptance_from_decimal_string(dec_str,
wrap_adjacency=args.wrap)
    if not report.ok:
        print(f"Acceptance check: {report.reason}", file=sys.stderr)
        return 1

    print(f"Accepted. Grid: {report.m} x {report.m} (tokens=
{len(report.indexes)})")

    try:
        qc, cc = derive_quantum_and_classical_from_grid(
            indexes=report.indexes,
            m=report.m,
            max_qubits=args.max_qubits,
            max_layers=args.max_layers,
            reversible_only=args.reversible_only,
        )
    except ModuleNotFoundError:
        print(
            "Missing dependency: qiskit.\n"
```

```
'Install with: pip install "qiskit[visualization]" matplotlib numpy
pillow',
        file=sys.stderr,
    )
    return 2

outdir = args.outdir
ensure_dir(outdir)

# Save text gate sequences
q_txt = os.path.join(outdir, "source_quantum.txt")
c_txt = os.path.join(outdir, "source_classical.txt")
write_gate_sequence(qc, q_txt)
write_gate_sequence(cc, c_txt)

wrote: List[str] = [q_txt, c_txt]

# Save PNGs (optional)
if not args.no_png:
    try:
        q_png = os.path.join(outdir, "quantum.png")
        c_png = os.path.join(outdir, "classical.png")
        a_png = os.path.join(outdir, "assembly.png")

        save_circuit_png(qc, q_png)
        save_circuit_png(cc, c_png)
        wrote.extend([q_png, c_png])

    try:
        combine_pngs_side_by_side(q_png, c_png, a_png)
        wrote.append(a_png)
    except ModuleNotFoundError:
        # Pillow missing; keep individual PNGs
        pass

except Exception as e:
    print(f"PNG render failed: {e}", file=sys.stderr)
    print("Tip: ensure matplotlib + pillow are installed.",
file=sys.stderr)

# Save a minimal manifest (handy for pipelines)
manifest = {
    "accepted": True,
    "grid_m": report.m,
    "token_count": len(report.indexes),
    "wrap_adjacency": bool(args.wrap),
    "reversible_only": bool(args.reversible_only),
    "max_qubits": int(args.max_qubits),
    "max_layers": int(args.max_layers),
    "outputs": wrote,
}
mf = os.path.join(outdir, "manifest.json")
with open(mf, "w", encoding="utf-8") as f:
    json.dump(manifest, f, indent=2)
```

```
wrote.append(mf)

print("Wrote:")
for p in wrote:
    print(f" {p}")

return 0


def cmd_menu(_: argparse.Namespace) -> int:
    # Simple interactive fallback (similar to original)
    print("\nC700h – Acceptance → Quantum/Classical Circuit Deriver\n")
    while True:
        try:
            mode = input("Enter 1=derive, 2=accept-only, 3=exit: ").strip()
        except (EOFError, KeyboardInterrupt):
            print()
            return 0

        if mode == "3":
            return 0

        if mode not in ("1", "2"):
            print("Unknown mode.\n")
            continue

        raw = input("Paste accepted state (decimal or 0b... binary): ").strip()
        wrap = input("Wrap-around adjacency? (y/n): ")
        ".strip().lower().startswith("y")"

        try:
            dec_str = parse_accepted_state_to_decimal_string(raw)
        except ValueError as e:
            print(str(e) + "\n")
            continue

        report = verify_acceptance_from_decimal_string(dec_str,
wrap_adjacency=wrap)
        print(f"\nAcceptance check: {report.reason}")
        if not report.ok:
            print("Not accepted.\n")
            continue

        print(f"Grid: {report.m} x {report.m} (tokens={len(report.indexes)})")

        if mode == "2":
            print()
            continue

        reversible_only = input("Reversible-only (exact classical correspondent)?\n(y/n): ").strip().lower().startswith("y")
        max_qubits = int(input("Max qubits (e.g. 8): ").strip() or "8")
        max_layers = int(input("Max layers (e.g. 16): ").strip() or "16")
        outdir = input("Output dir (default: out): ").strip() or "out"
```

```
try:
    qc, cc = derive_quantum_and_classical_from_grid(
        indexes=report.indexes,
        m=report.m,
        max_qubits=max_qubits,
        max_layers=max_layers,
        reversible_only=reversible_only,
    )
except ModuleNotFoundError:
    print(
        '\nMissing dependency. Install with:\n  pip install
"qiskit[visualization]" matplotlib numpy pillow\n'
    )
    continue

ensure_dir(outdir)
write_gate_sequence(qc, os.path.join(outdir, "source_quantum.txt"))
write_gate_sequence(cc, os.path.join(outdir, "source_classical.txt"))

try:
    qpng = os.path.join(outdir, "quantum.png")
    cpng = os.path.join(outdir, "classical.png")
    apng = os.path.join(outdir, "assembly.png")
    save_circuit_png(qc, qpng)
    save_circuit_png(cc, cpng)
    try:
        combine_pngs_side_by_side(qpng, cpng, apng)
    except ModuleNotFoundError:
        pass

    print("\nWrote:")
    print(f"  {qpng}")
    print(f"  {cpng}")
    if os.path.exists(apng):
        print(f"  {apng}")
    print(f"  {os.path.join(outdir, 'source_quantum.txt')}")
    print(f"  {os.path.join(outdir, 'source_classical.txt')}\n")
except Exception as e:
    print(f"\nRender failed: {e}\n")

# -----
# CLI wiring
# -----


def build_parser() -> argparse.ArgumentParser:
    p = argparse.ArgumentParser(
        prog="C700h.py",
        description="C700 Host: acceptance + deterministic quantum/classical
circuit derivation",
    )
    sub = p.add_subparsers(dest="cmd", required=False)
```

```
# accept
pa = sub.add_parser("accept", help="Run acceptance check only")
pa.add_argument("--state", type=str, default=None, help="Accepted state as decimal or 0b... binary")
pa.add_argument("--in", dest="infile", type=str, default=None, help="Read accepted state from a text file")
pa.add_argument("--wrap", action="store_true", help="Enable wrap-around adjacency checks")
pa.add_argument("--json", action="store_true", help="Emit JSON report")
pa.add_argument("--show-indexes", action="store_true", help="Print decoded 7-digit token indexes")
pa.add_argument("--show-hex", action="store_true", help="Print hex colors for each token")
pa.set_defaults(func=cmd_accept)

# derive
pd = sub.add_parser("derive", help="Accept + derive circuits + write outputs")
pd.add_argument("--state", type=str, default=None, help="Accepted state as decimal or 0b... binary")
pd.add_argument("--in", dest="infile", type=str, default=None, help="Read accepted state from a text file")
pd.add_argument("--wrap", action="store_true", help="Enable wrap-around adjacency checks")
pd.add_argument("--reversible-only", action="store_true", help="Force reversible-only mapping (exact classical correspondent)")
pd.add_argument("--max-qubits", type=int, default=8, help="Maximum qubits (default: 8)")
pd.add_argument("--max-layers", type=int, default=16, help="Maximum layers (default: 16)")
pd.add_argument("--outdir", type=str, default="out", help="Output directory (default: out)")
pd.add_argument("--no-png", action="store_true", help="Skip PNG rendering (write text outputs only)")
pd.set_defaults(func=cmd_derive)

# menu
pm = sub.add_parser("menu", help="Interactive menu mode (fallback)")
pm.set_defaults(func=cmd_menu)

return p

def main(argv: Optional[Sequence[str]] = None) -> int:
    parser = build_parser()
    args = parser.parse_args(argv)

    # If no subcommand provided, default to menu for convenience
    if not getattr(args, "cmd", None):
        return cmd_menu(args)

    # Require one of --state / --in for accept/derive
    if args.cmd in ("accept", "derive") and not args.state and not args.infile:
        parser.error("accept/derive require --state or --in")
```

```
    return int(args.func(args))
```

```
if __name__ == "__main__":
    raise SystemExit(main())
```

## [64] Hardware/C700h\_updated.md

- **Bytes:** 1946
- **Type:** text

Done – I repurposed the same **accept → deterministic-derive → write outputs + manifest** automation pattern in your host utility, but made the **Category be semantic** (not “one node per cell”).

**### Semantic meaning used (deterministic)**

For an accepted integer-state:

\* **Objects** = the **unique semantic symbols** induced by the same mapping you use in `derive`:

- \* token → `gate = GATES[token % len(GATES)]`
- \* rotation gates become `rx(k=...) / ry(k=...) / rz(k=...)` (discrete angle class)
- \* `cx` becomes `cx(shift=...)` (uses the same `q=min(m,max\_qubits)` window)
- \* `--reversible-only` coerces semantics to **x/cx only** exactly like `derive`
- \* **Generating morphisms** = **observed orthogonal adjacency** between semantic symbols inside the same `layers × q` window used by the circuit derivation:

- \* East edges (`Excount`) and South edges (`Sxcount`), aggregated with counts + a few example positions.
- \* The resulting category is the **free category on that generator graph** (identities + composition-by-paths implicit).

**### Outputs (`category` command)**

Writes to `--outdir`:

- \* `category.json` (semantic category presentation)
- \* `grid.png` (token-color grid)
- \* `category.png` (semantic object-relation diagram)
- \* `category\_assembly.png` (side-by-side grid + category, if `--assembly`)
- \* `category\_manifest.json`

**### Download**

[Download the updated C700h.py]([sandbox:/mnt/data/C700h\\_updated.py](#))

**### How to run**

```
```powershell
# JSON only
python C700h_updated.py category --state "1234567..." --outdir out_cat --no-png

# With PNGs + assembly
pip install matplotlib pillow
python C700h_updated.py category --state "0b101010..." --outdir out_cat --assembly
````
```

If you instead want “semantic objects” to be **\*\*unique tokens/colors\*\*** (not gate-semantics), tell me and I’ll switch the object definition to token-IDs while keeping the same pipeline + outputs.

## [65] Hardware/C700h\_updated.py

- **Bytes:** 36123
- **Type:** text

```
#!/usr/bin/env python3
"""
C700h.py (Python 3.14.2)

C700 Host utility: Acceptance → Deterministic Quantum/Classical Circuit
Derivation.

Rewritten from the uploaded dimension-generator.py:
- Same acceptance rules (7-digit tokenization, perfect-square grid, color range,
adjacency conflict).
- Same deterministic mapping into Qiskit circuits and optional classical "shadow".
- Adds a clean CLI with subcommands: accept, derive, menu.
- Still supports huge integer accepted-states (decimal) and binary "0b...".

Dependencies:
- derive PNG output: pip install "qiskit[visualization]" matplotlib numpy
pillow
- category PNG output: pip install matplotlib pillow

Notes:
- Qiskit is imported lazily so `accept` works without Qiskit installed.
- Output defaults to ./out
"""

from __future__ import annotations

import argparse
import json
import math
import os
import re
import sys
```

```
from dataclasses import dataclass
from typing import Any, Iterable, List, Optional, Sequence, Tuple

# -----
# Constants
# -----


SEGMENT_LEN: int = 7
MAX_COLOR_INDEX: int = 16 ** 6 # 16777216

GATES: List[str] = ["x", "y", "z", "h", "s", "sdg", "t", "tdg", "rx", "ry", "rz",
"cx"]


# -----
# Acceptance helpers
# -----


def is_perfect_square(n: int) -> bool:
    if n <= 0:
        return False
    r = math.sqrt(n)
    return r * r == n


def decode_id_to_color_indexes(id_string: str, segment_length: int = SEGMENT_LEN)
-> List[int]:
    """
    Mirrors map.js behavior:
    - Split left-to-right into fixed segment_length chunks.
    - No padding.
    - Last segment may be shorter.
    - Non-integer segments are skipped.
    """
    out: List[int] = []
    for i in range(0, len(id_string), segment_length):
        seg = id_string[i : i + segment_length]
        try:
            out.append(int(seg, 10))
        except ValueError:
            # keep behavior: silently ignore invalid chunks
            pass
    return out


def are_valid_color_indexes(indexes: Sequence[int]) -> bool:
    return all(0 <= x <= MAX_COLOR_INDEX for x in indexes)


def has_adjacent_conflict(indexes: Sequence[int], wrap: bool = False) -> bool:
    """
    True if:
    - token count not perfect square, OR
    """
```

```
- any orthogonal neighbor pair in the grid is equal.
Optional wrap-around adjacency on right and bottom edges.
"""
if not is_perfect_square(len(indexes)):
    return True

m = math.sqrt(len(indexes))

def idx(r: int, c: int) -> int:
    return indexes[r * m + c]

for r in range(m):
    for c in range(m):
        cur = idx(r, c)

        # right
        if c + 1 < m:
            if cur == idx(r, c + 1):
                return True
        elif wrap and m > 1:
            if cur == idx(r, 0):
                return True

        # down
        if r + 1 < m:
            if cur == idx(r + 1, c):
                return True
        elif wrap and m > 1:
            if cur == idx(0, c):
                return True

return False

def color_index_to_hex(index: int) -> str:
    # 1 -> #000000, 16^6 -> #FFFFFF
    v = index - 1
    return "#" + format(v, "06x")

@dataclass(frozen=True)
class AcceptanceReport:
    ok: bool
    reason: str
    m: int
    indexes: List[int]
    hex_colors: List[str]

    def to_dict(self) -> dict[str, Any]:
        return {
            "ok": self.ok,
            "reason": self.reason,
            "m": self.m,
            "token_count": len(self.indexes),
```

```
        "indexes": self.indexes,
        "hex_colors": self.hex_colors,
    }

def verify_acceptance_from_decimal_string(id_decimal: str, wrap_adjacency: bool = False) -> AcceptanceReport:
    indexes = decode_id_to_color_indexes(id_decimal, segment_length=SEGMENT_LEN)

    if not is_perfect_square(len(indexes)):
        return AcceptanceReport(
            False,
            f"Token count {len(indexes)} is not a perfect square.",
            0,
            list(indexes),
            [],
        )

    m = math.sqrt(len(indexes))

    if not are_valid_color_indexes(indexes):
        return AcceptanceReport(
            False,
            "One or more tokens are outside [1..16^6].",
            m,
            list(indexes),
            [],
        )

    if has_adjacent_conflict(indexes, wrap=wrap_adjacency):
        return AcceptanceReport(
            False,
            "Adjacency conflict: at least one orthogonal neighbor pair is equal.",
            m,
            list(indexes),
            [],
        )

    hex_colors = [color_index_to_hex(x) for x in indexes]
    return AcceptanceReport(True, "Accepted.", m, list(indexes), hex_colors)

# -----
# Deterministic circuit derivation
# -----


def token_to_angle(token: int) -> float:
    # Deterministic discrete angle: multiples of pi/16 (never 0)
    k = (token % 32) + 1
    return k * (math.pi / 16.0)


def derive_quantum_and_classical_from_grid(
    indexes: Sequence[int],
```

```
m: int,
max_qubits: int = 8,
max_layers: int = 16,
reversible_only: bool = False,
):
"""
Deterministic mapping:
- Qubits = min(m, max_qubits)
- Layers = min(m, max_layers)
- Gate for (r,c) chosen by token % len(GATES)

classical_shadow keeps:
- x and cx when they occur (and in reversible_only mode)
"""

# Lazy import so accept/help can run without qiskit installed
from qiskit import QuantumCircuit # type: ignore

q = min(m, max_qubits)
layers = min(m, max_layers)

qc = QuantumCircuit(q, name="derived_quantum")
cc = QuantumCircuit(q, name="classical_shadow")

def grid_token(r: int, c: int) -> int:
    return indexes[r * m + c]

for r in range(layers):
    for c in range(q):
        tok = grid_token(r, c)
        gate = GATES[tok % len(GATES)]

        if reversible_only:
            # force into exact classical correspondence
            gate = "cx" if (tok % 2 == 1 and q > 1) else "x"

        if gate in ("x", "y", "z", "h", "s", "sdg", "t", "tdg"):
            getattr(qc, gate)(c)
            if gate == "x":
                cc.x(c)

        elif gate == "cx":
            if q == 1:
                qc.x(c)
                cc.x(c)
            else:
                control = c
                shift = 1 + (tok % (q - 1))
                target = (c + shift) % q
                qc.cx(control, target)
                cc.cx(control, target)

        elif gate in ("rx", "ry", "rz"):
            ang = token_to_angle(tok)
            getattr(qc, gate)(ang, c)
```

```
# no classical equivalent in shadow

qc.barrier()

return qc, cc

# -----
# Output helpers
# -----


def ensure_dir(path: str) -> None:
    os.makedirs(path, exist_ok=True)

def save_circuit_png(qc, path: str) -> None:
    from qiskit.visualization import circuit_drawer # type: ignore
    circuit_drawer(qc, output="mpl", filename=path)

def combine_pngs_side_by_side(left_path: str, right_path: str, out_path: str) ->
None:
    """
    Create a clean side-by-side "assembly" PNG.

    Improvements over the earlier helper:
    - trims excess whitespace around each image (matplotlib outputs often have
    large margins)
    - matches heights by resizing proportionally
    - uses a white background (no black blocks)
    - vertically centers both panels

    Requires: Pillow
    """
    from PIL import Image, ImageChops # type: ignore

    def _trim_whitespace(im: Image.Image, bg_rgb=(255, 255, 255), pad: int = 16) -
> Image.Image:
        # Work in RGB for robust diff.
        rgb = im.convert("RGB")
        bg = Image.new("RGB", rgb.size, bg_rgb)
        diff = ImageChops.difference(rgb, bg).convert("L")
        # Threshold small compression artifacts
        diff = diff.point(lambda p: 255 if p > 10 else 0)
        bbox = diff.getbbox()
        if not bbox:
            return im
        x0, y0, x1, y1 = bbox
        x0 = max(0, x0 - pad)
        y0 = max(0, y0 - pad)
        x1 = min(im.width, x1 + pad)
        y1 = min(im.height, y1 + pad)
        return im.crop((x0, y0, x1, y1))
```

```
a = Image.open(left_path).convert("RGBA")
b = Image.open(right_path).convert("RGBA")

a = _trim_whitespace(a)
b = _trim_whitespace(b)

# Match heights (like the quantum/classical assembly)
target_h = max(a.height, b.height)

def _resize_to_h(im: Image.Image, h: int) -> Image.Image:
    if im.height == h:
        return im
    w = int(round(im.width * (h / float(im.height))))
    return im.resize((w, h), resample=Image.Resampling.LANCZOS)

a2 = _resize_to_h(a, target_h)
b2 = _resize_to_h(b, target_h)

pad = 24
w = a2.width + b2.width + pad * 3
h = target_h + pad * 2

canvas = Image.new("RGBA", (w, h), (255, 255, 255, 255))

# Center vertically (mostly redundant after height-match, but keeps future-proof)
y_a = pad + (target_h - a2.height) // 2
y_b = pad + (target_h - b2.height) // 2

canvas.paste(a2, (pad, y_a), a2)
canvas.paste(b2, (pad * 2 + a2.width, y_b), b2)

# Save as RGB (avoids odd alpha rendering in some viewers)
canvas.convert("RGB").save(out_path)

def write_gate_sequence(qc, out_txt: str) -> None:
    """
    Qiskit 1.x compatible:
    - qc.data yields CircuitInstruction objects
    - Qubit index obtained via qc.find_bit(qubit).index
    """
    parts: List[str] = []

    for ci in qc.data:
        op = ci.operation
        if op.name == "barrier":
            continue

        qinds = [qc.find_bit(q).index for q in ci.qubits]

        if op.name in ("rx", "ry", "rz"):
            angle = op.params[0]
            parts.append(f"{op.name}({angle},{qinds[0]})")
        elif op.name == "cx":
```

```
        parts.append(f"cx({qinds[0]},{qinds[1]})")
    else:
        parts.append(f"{op.name}({qinds[0]})")

    with open(out_txt, "w", encoding="utf-8") as f:
        f.write(" ".join(parts) + "\n")

# -----
# Deterministic semantic category derivation + diagram rendering
# -----


def _hex_luma(hex_color: str) -> float:
    """Return perceived luminance in [0,1] for '#RRGGBB'."""
    hc = hex_color.lstrip("#")
    r = int(hc[0:2], 16) / 255.0
    g = int(hc[2:4], 16) / 255.0
    b = int(hc[4:6], 16) / 255.0
    return 0.2126 * r + 0.7152 * g + 0.0722 * b


def _hex_to_rgb01(hex_color: str) -> tuple[float, float, float]:
    hc = hex_color.lstrip("#")
    r = int(hc[0:2], 16) / 255.0
    g = int(hc[2:4], 16) / 255.0
    b = int(hc[4:6], 16) / 255.0
    return (r, g, b)


def token_to_semantic_symbol(token: int, q: int, reversible_only: bool = False) ->
str:
    """
    Semantic label for a token.

    - Mirrors the derive mapping: gate name chosen by token % len(GATES)
    - In reversible_only: coerces to x/cx exactly like circuit derivation
    - For rotation gates: includes a discrete angle class (k·pi/16)
    - For cx: includes deterministic shift class when q>1
    """
    gate = GATES[token % len(GATES)]
    if reversible_only:
        gate = "cx" if (token % 2 == 1 and q > 1) else "x"

    if gate in ("rx", "ry", "rz"):
        k = (token % 32) + 1
        return f"{gate}(k={k})"
    if gate == "cx":
        if q <= 1:
            return "x"
        shift = 1 + (token % (q - 1))
        return f"cx(shift={shift})"
    return gate
```

```
def derive_semantic_category_from_grid(
    indexes: Sequence[int],
    m: int,
    max_qubits: int = 8,
    max_layers: int = 16,
    reversible_only: bool = False,
) -> dict[str, Any]:
    """
    Build a deterministic *semantic* category presentation.

    Semantics (default):
        - Objects = unique semantic symbols induced by the derive mapping (gate
        semantics),
        rather than one object per cell.
        - Generators = observed orthogonal adjacency relations between semantic
        symbols
        (east + south) restricted to the same window used by circuit derivation:
        q = min(m, max_qubits)
        layers = min(m, max_layers)

    The category is presented as the free category on this generator graph
    (identities + composition by paths are implicit).

    Returns a JSON-serializable dict with objects + aggregated generators.
    """
    q = min(m, max_qubits)
    layers = min(m, max_layers)

    def grid_token(r: int, c: int) -> int:
        return int(indexes[r * m + c])

    # Aggregate objects (semantic symbols) and representative/average colors
    obj_set: set[str] = set()
    rgb_sum: dict[str, list[float]] = {}
    rgb_n: dict[str, int] = {}

    # Aggregate generators: key=(src,dst,kind) -> {count, examples}
    gen: dict[tuple[str, str, str], dict[str, Any]] = {}

    def add_obj(sym: str, hex_color: str) -> None:
        obj_set.add(sym)
        r, g, b = _hex_to_rgb01(hex_color)
        if sym not in rgb_sum:
            rgb_sum[sym] = [0.0, 0.0, 0.0]
            rgb_n[sym] = 0
        rgb_sum[sym][0] += r
        rgb_sum[sym][1] += g
        rgb_sum[sym][2] += b
        rgb_n[sym] += 1

    def add_edge(src: str, dst: str, kind: str, at_rc: tuple[int, int]) -> None:
        key = (src, dst, kind)
        if key not in gen:
```

```
        gen[key] = {"src": src, "dst": dst, "kind": kind, "count": 0,
"examples": []}
        gen[key]["count"] += 1
        # store up to a few examples deterministically
        if len(gen[key]["examples"]) < 12:
            gen[key]["examples"].append({"at": [at_rc[0], at_rc[1]]})

    # Walk same window used for circuit derivation (layers x q)
    for r in range(layers):
        for c in range(q):
            tok = grid_token(r, c)
            sym = token_to_semantic_symbol(tok, q=q,
reversible_only=reversible_only)
            add_obj(sym, color_index_to_hex(tok))

            # East adjacency (within window)
            if c + 1 < q:
                tok2 = grid_token(r, c + 1)
                sym2 = token_to_semantic_symbol(tok2, q=q,
reversible_only=reversible_only)
                add_obj(sym2, color_index_to_hex(tok2))
                add_edge(sym, sym2, "E", (r, c))

            # South adjacency (within window)
            if r + 1 < layers:
                tok2 = grid_token(r + 1, c)
                sym2 = token_to_semantic_symbol(tok2, q=q,
reversible_only=reversible_only)
                add_obj(sym2, color_index_to_hex(tok2))
                add_edge(sym, sym2, "S", (r, c))

    objects = sorted(obj_set)

    # Representative color per object = average of contributing token colors
    obj_color: dict[str, str] = {}
    for o in objects:
        n = max(1, int(rgb_n.get(o, 1)))
        r, g, b = (rgb_sum[o][0] / n, rgb_sum[o][1] / n, rgb_sum[o][2] / n)
        obj_color[o] = "#" + "".join(f"{{int(max(0,min(255,round(v*255))))}}:02x}" for v in (r, g, b))

    generators = []
    for (src, dst, kind), payload in gen.items():
        label = f"{kind}x{payload['count']}"
        generators.append(
            {
                "src": src,
                "dst": dst,
                "kind": kind,
                "count": int(payload["count"]),
                "label": label,
                "examples": payload["examples"],
            }
        )
```

```
# stable ordering for determinism
generators.sort(key=lambda e: (e["src"], e["dst"], e["kind"]))

return {
    "kind": "semantic_free_category_presentation",
    "objects": [{"id": o, "color": obj_color[o]} for o in objects],
    "generators": generators,
    "composition": "paths (free category on the generator graph)",
    "identities": "implicit for each object",
    "window": {"q": q, "layers": layers, "m": m},
    "reversible_only": bool(reversible_only),
}
}

def render_grid_png(
    indexes: Sequence[int],
    hex_colors: Sequence[str],
    m: int,
    out_path: str,
    show_cell_labels: bool = False,
) -> None:
    """Render the token grid as a deterministic PNG."""
    try:
        import matplotlib.pyplot as plt
        from matplotlib.patches import Rectangle
    except ModuleNotFoundError as e:
        raise ModuleNotFoundError(
            "Missing dependency: matplotlib (needed for grid/category PNG)."
        )
    print("Install with: pip install matplotlib")
    ) from e

    fig_w = max(4.0, float(m) * 1.05)
    fig_h = max(4.0, float(m) * 1.05)
    fig, ax = plt.subplots(figsize=(fig_w, fig_h))

    for r in range(m):
        for c in range(m):
            i = r * m + c
            fc = hex_colors[i]
            ax.add_patch(Rectangle((c, -r - 1), 1, 1, facecolor=fc,
edgecolor="black", linewidth=1.0))
            if show_cell_labels:
                luma = _hex_luma(fc)
                txt_color = "white" if luma < 0.45 else "black"
                ax.text(c + 0.5, -r - 0.5, str(indexes[i]), ha="center",
va="center", fontsize=6, color=txt_color)

    ax.set_aspect("equal")
    ax.set_xlim(0, m)
    ax.set_ylim(-m, 0)
    ax.axis("off")
    fig.tight_layout()
    fig.savefig(out_path, dpi=220)
```

```
plt.close(fig)

def render_semantic_category_png(category: dict[str, Any], out_path: str,
show_edge_labels: bool = True) -> None:
    """
    Render the semantic category diagram as a deterministic PNG.

    Key goals:
    - stable (deterministic) layout
    - readable (avoid massive whitespace / "content shoved into a corner")
    - assembly-friendly (cropped margins)

    Layout:
    - nodes placed on a circle in sorted object-id order (deterministic)
    - edges drawn as directed arrows
    - node labels are shortened to fit inside nodes (full ids remain in
category.json)
        - edge labels are only shown when count > 1 (or if you pass --hide-edge-
labels to hide all)
    """
    try:
        import matplotlib.pyplot as plt
        from matplotlib.patches import FancyArrowPatch, Circle
    except ModuleNotFoundError as e:
        raise ModuleNotFoundError(
            "Missing dependency: matplotlib (needed for category PNG). Install
with: pip install matplotlib"
        ) from e

    objects = [o["id"] for o in category.get("objects", [])]
    colors = {o["id"]: o.get("color", "#888888") for o in category.get("objects",
[])}
    gens = list(category.get("generators", []))

    # Deterministic circle layout
    n = max(1, len(objects))
    import math as _math

    pos: dict[str, tuple[float, float]] = {}
    for i, oid in enumerate(objects):
        ang = (2.0 * _math.pi * i) / n
        pos[oid] = (_math.cos(ang), _math.sin(ang))

    # Short labels so text doesn't blow up layout
    def _short_label(oid: str) -> str:
        if oid.startswith("rx(k=") or oid.startswith("ry(k=") or
oid.startswith("rz(k="):
            # rx(k=25) -> rx25
            m = re.match(r"^(r[xyz])\((k=(\d+))\)$", oid)
            if m:
                return f"{m.group(1)}{m.group(2)}"
        if oid.startswith("cx(shift="):
            m = re.match(r"^(cx\((shift=(\d+))\)$", oid)
```

```
if m:
    return f"cx{m.group(1)}"
return oid

# Fixed figure size (avoid n-based runaway whitespace)
fig, ax = plt.subplots(figsize=(8.0, 8.0))

idx_map = {oid: i for i, oid in enumerate(objects)}

def _curve_for(kind: str, src: str, dst: str) -> float:
    if src == dst:
        return 0.35
    base = 0.18 if kind == "E" else -0.18
    base += ((idx_map[src] - idx_map[dst]) % 7 - 3) * 0.01
    return base

# Draw edges first
for e in gens:
    src = e.get("src")
    dst = e.get("dst")
    if src not in pos or dst not in pos:
        continue
    x0, y0 = pos[src]
    x1, y1 = pos[dst]
    kind = str(e.get("kind", ""))
    rad = _curve_for(kind, src, dst)

    arrow = FancyArrowPatch(
        (x0, y0),
        (x1, y1),
        arrowstyle="->",
        mutation_scale=12,
        linewidth=1.15,
        color="black",
        connectionstyle=f"arc3,rad={rad}",
    )
    ax.add_patch(arrow)

    if show_edge_labels:
        # reduce clutter: only label multiplicities > 1
        count = int(e.get("count", 1) or 1)
        if count > 1:
            mx, my = (x0 + x1) / 2.0, (y0 + y1) / 2.0
            ax.text(mx, my + rad * 0.35, f"{kind}x{count}", ha="center",
                    va="center", fontsize=9)

# Draw nodes
radius = 0.13 # stable size
for oid in objects:
    x, y = pos[oid]
    fc = colors.get(oid, "#888888")
    ax.add_patch(Circle((x, y), radius=radius, facecolor=fc,
edgecolor="black", linewidth=1.2))
    luma = _hex_luma(fc)
```

```
        txt_color = "white" if luma < 0.45 else "black"
        ax.text(x, y, _short_label(oid), ha="center", va="center", fontsize=10,
color=txt_color)

        # FIXED limits so content stays centered and uses the canvas
        ax.set_aspect("equal")
        ax.set_xlim(-1.35, 1.35)
        ax.set_ylim(-1.35, 1.35)
        ax.axis("off")

        # Save tightly (removes massive margins) but with padding for labels/arrows
        fig.savefig(out_path, dpi=220, bbox_inches="tight", pad_inches=0.2,
facecolor="white")
        plt.close(fig)

# -----
# Input parsing
# -----


_DEC_RE = re.compile(r"^[0-9]+$")

def parse_accepted_state_to_decimal_string(raw: str) -> str:
    """
    Accepts:
    - decimal integer string (possibly huge)
    - binary with 0b prefix
    Returns canonical decimal string (no leading zeros except "0").
    """
    s = raw.strip()
    if s.lower().startswith("0b"):
        n = int(s, 2)
        return str(n)

    if not _DEC_RE.fullmatch(s):
        raise ValueError("Not a valid decimal integer string (or 0b... binary).")

    # allow huge integers; normalize leading zeros
    s2 = s.lstrip("0")
    return s2 if s2 != "" else "0"

def read_text_file(path: str) -> str:
    with open(path, "r", encoding="utf-8") as f:
        return f.read().strip()

# -----
# Commands
# -----


def cmd_accept(args: argparse.Namespace) -> int:
    raw = read_text_file(args.infile) if args.infile else args.state
```

```
if raw is None:
    print("No state provided. Use --state or --in.", file=sys.stderr)
    return 2

try:
    dec_str = parse_accepted_state_to_decimal_string(raw)
except ValueError as e:
    print(str(e), file=sys.stderr)
    return 2

report = verify_acceptance_from_decimal_string(dec_str,
wrap_adjacency=args.wrap)
if args.json:
    print(json.dumps(report.to_dict(), indent=2))
else:
    print(report.reason)
    if report.ok:
        print(f"Grid: {report.m} x {report.m} (tokens={len(report.indexes)})")
        if args.show_hex:
            print("Hex colors:")
            print(" ".join(report.hex_colors))
        if args.show_indexes:
            print("Indexes:")
            print(" ".join(str(x) for x in report.indexes))

return 0 if report.ok else 1

def cmd_derive(args: argparse.Namespace) -> int:
    raw = read_text_file(args.infile) if args.infile else args.state
    if raw is None:
        print("No state provided. Use --state or --in.", file=sys.stderr)
        return 2

    try:
        dec_str = parse_accepted_state_to_decimal_string(raw)
    except ValueError as e:
        print(str(e), file=sys.stderr)
        return 2

    report = verify_acceptance_from_decimal_string(dec_str,
wrap_adjacency=args.wrap)
    if not report.ok:
        print(f"Acceptance check: {report.reason}", file=sys.stderr)
        return 1

    print(f"Accepted. Grid: {report.m} x {report.m} (tokens=
{len(report.indexes)})")

    try:
        qc, cc = derive_quantum_and_classical_from_grid(
            indexes=report.indexes,
            m=report.m,
            max_qubits=args.max_qubits,
```

```
        max_layers=args.max_layers,
        reversible_only=args.reverseable_only,
    )
except ModuleNotFoundError:
    print(
        "Missing dependency: qiskit.\n"
        'Install with: pip install "qiskit[visualization]" matplotlib numpy
pillow',
        file=sys.stderr,
    )
    return 2

outdir = args.outdir
ensure_dir(outdir)

# Save text gate sequences
q_txt = os.path.join(outdir, "source_quantum.txt")
c_txt = os.path.join(outdir, "source_classical.txt")
write_gate_sequence(qc, q_txt)
write_gate_sequence(cc, c_txt)

wrote: List[str] = [q_txt, c_txt]

# Save PNGs (optional)
if not args.no_png:
    try:
        q_png = os.path.join(outdir, "quantum.png")
        c_png = os.path.join(outdir, "classical.png")
        a_png = os.path.join(outdir, "assembly.png")

        save_circuit_png(qc, q_png)
        save_circuit_png(cc, c_png)
        wrote.extend([q_png, c_png])

    try:
        combine_pngs_side_by_side(q_png, c_png, a_png)
        wrote.append(a_png)
    except ModuleNotFoundError:
        # Pillow missing; keep individual PNGs
        pass

    except Exception as e:
        print(f"PNG render failed: {e}", file=sys.stderr)
        print("Tip: ensure matplotlib + pillow are installed.",
file=sys.stderr)

# Save a minimal manifest (handy for pipelines)
manifest = {
    "accepted": True,
    "grid_m": report.m,
    "token_count": len(report.indexes),
    "wrap_adjacency": bool(args.wrap),
    "reverseable_only": bool(args.reverseable_only),
    "max_qubits": int(args.max_qubits),
```

```
"max_layers": int(args.max_layers),
"outputs": wrote,
}
mf = os.path.join(outdir, "manifest.json")
with open(mf, "w", encoding="utf-8") as f:
    json.dump(manifest, f, indent=2)
wrote.append(mf)

print("Wrote:")
for p in wrote:
    print(f" {p}")

return 0

def cmd_category(args: argparse.Namespace) -> int:
    raw = read_text_file(args.infile) if args.infile else args.state
    if raw is None:
        print("No state provided. Use --state or --in.", file=sys.stderr)
        return 2

    try:
        dec_str = parse_accepted_state_to_decimal_string(raw)
    except ValueError as e:
        print(str(e), file=sys.stderr)
        return 2

    report = verify_acceptance_from_decimal_string(dec_str,
wrap_adjacency=args.wrap)
    if not report.ok:
        print(f"Acceptance check: {report.reason}", file=sys.stderr)
        return 1

    outdir = args.outdir
    ensure_dir(outdir)

    # Derive semantic category (free category presentation over semantic adjacency
graph)
    cat = derive_semantic_category_from_grid(
        indexes=report.indexes,
        m=report.m,
        max_qubits=args.max_qubits,
        max_layers=args.max_layers,
        reversible_only=args.reversible_only,
    )

    cat_path = os.path.join(outdir, "category.json")
    with open(cat_path, "w", encoding="utf-8") as f:
        json.dump(cat, f, indent=2)

    wrote: List[str] = [cat_path]

    # PNG outputs (optional)
    if not args.no_png:
        # grid view (helps as a deterministic "assembly" companion)
```

```
grid_png = os.path.join(outdir, "grid.png")
cat_png = os.path.join(outdir, "category.png")
asm_png = os.path.join(outdir, "category_assembly.png")

try:
    render_grid_png(
        indexes=report.indexes,
        hex_colors=report.hex_colors,
        m=report.m,
        out_path=grid_png,
        show_cell_labels=args.show_grid_labels,
    )
    wrote.append(grid_png)
except Exception as e:
    print(f"Grid PNG render failed: {e}", file=sys.stderr)

try:
    render_semantic_category_png(cat, cat_png, show_edge_labels=not
args.hide_edge_labels)
    wrote.append(cat_png)
except Exception as e:
    print(f"Category PNG render failed: {e}", file=sys.stderr)

if args.assembly:
    try:
        if os.path.exists(grid_png) and os.path.exists(cat_png):
            combine_pngs_side_by_side(grid_png, cat_png, asm_png)
            wrote.append(asm_png)
    except ModuleNotFoundError:
        # Pillow missing; assembly is optional
        pass
    except Exception:
        pass

manifest = {
    "accepted": True,
    "grid_m": report.m,
    "token_count": len(report.indexes),
    "wrap_adjacency": bool(args.wrap),
    "reversible_only": bool(args.reversible_only),
    "max_qubits": int(args.max_qubits),
    "max_layers": int(args.max_layers),
    "outputs": wrote,
    "category_kind": cat.get("kind", "semantic_free_category_presentation"),
}
mf = os.path.join(outdir, "category_manifest.json")
with open(mf, "w", encoding="utf-8") as f:
    json.dump(manifest, f, indent=2)
wrote.append(mf)

print("Wrote:")
for p in wrote:
    print(f"  {p}")
return 0
```

```
def cmd_menu(_: argparse.Namespace) -> int:
    # Simple interactive fallback (similar to original)
    print("\nC700h – Acceptance → Quantum/Classical Circuit Deriver\n")
    while True:
        try:
            mode = input("Enter 1=derive, 2=accept-only, 3=exit: ").strip()
        except (EOFError, KeyboardInterrupt):
            print()
            return 0

        if mode == "3":
            return 0

        if mode not in ("1", "2"):
            print("Unknown mode.\n")
            continue

        raw = input("Paste accepted state (decimal or 0b... binary): ").strip()
        wrap = input("Wrap-around adjacency? (y/n): ")
        ".strip().lower().startswith("y")"

        try:
            dec_str = parse_accepted_state_to_decimal_string(raw)
        except ValueError as e:
            print(str(e) + "\n")
            continue

        report = verify_acceptance_from_decimal_string(dec_str,
wrap_adjacency=wrap)
        print(f"\nAcceptance check: {report.reason}")
        if not report.ok:
            print("Not accepted.\n")
            continue

        print(f"Grid: {report.m} x {report.m} (tokens={len(report.indexes)})")

        if mode == "2":
            print()
            continue

        reversible_only = input("Reversible-only (exact classical correspondent)?\n(y/n): ").strip().lower().startswith("y")
        max_qubits = int(input("Max qubits (e.g. 8): ").strip() or "8")
        max_layers = int(input("Max layers (e.g. 16): ").strip() or "16")
        outdir = input("Output dir (default: out): ").strip() or "out"

        try:
            qc, cc = derive_quantum_and_classical_from_grid(
                indexes=report.indexes,
                m=report.m,
```

```
        max_qubits=max_qubits,
        max_layers=max_layers,
        reversible_only=reversible_only,
    )
except ModuleNotFoundError:
    print(
        '\nMissing dependency. Install with:\n  pip install
"qiskit[visualization]" matplotlib numpy pillow\n'
    )
    continue

ensure_dir(outdir)
write_gate_sequence(qc, os.path.join(outdir, "source_quantum.txt"))
write_gate_sequence(cc, os.path.join(outdir, "source_classical.txt"))

try:
    qpng = os.path.join(outdir, "quantum.png")
    cpng = os.path.join(outdir, "classical.png")
    apng = os.path.join(outdir, "assembly.png")
    save_circuit_png(qc, qpng)
    save_circuit_png(cc, cpng)
    try:
        combine_pngs_side_by_side(qpng, cpng, apng)
    except ModuleNotFoundError:
        pass

    print("\nWrote:")
    print(f"  {qpng}")
    print(f"  {cpng}")
    if os.path.exists(apng):
        print(f"  {apng}")
    print(f"  {os.path.join(outdir, 'source_quantum.txt')}")
    print(f"  {os.path.join(outdir, 'source_classical.txt')}\n")
except Exception as e:
    print(f"\nRender failed: {e}\n")

# -----
# CLI wiring
# -----


def build_parser() -> argparse.ArgumentParser:
    p = argparse.ArgumentParser(
        prog="C700h.py",
        description="C700 Host: acceptance + deterministic quantum/classical
circuit derivation",
    )
    sub = p.add_subparsers(dest="cmd", required=False)

    # accept
    pa = sub.add_parser("accept", help="Run acceptance check only")
    pa.add_argument("--state", type=str, default=None, help="Accepted state as
decimal or 0b... binary")
    pa.add_argument("--in", dest="infile", type=str, default=None, help="Read
```

```
accepted state from a text file")
    pa.add_argument("--wrap", action="store_true", help="Enable wrap-around
adjacency checks")
    pa.add_argument("--json", action="store_true", help="Emit JSON report")
    pa.add_argument("--show-indexes", action="store_true", help="Print decoded 7-
digit token indexes")
    pa.add_argument("--show-hex", action="store_true", help="Print hex colors for
each token")
    pa.set_defaults(func=cmd_accept)

    # derive
    pd = sub.add_parser("derive", help="Accept + derive circuits + write outputs")
    pd.add_argument("--state", type=str, default=None, help="Accepted state as
decimal or 0b... binary")
    pd.add_argument("--in", dest="infile", type=str, default=None, help="Read
accepted state from a text file")
    pd.add_argument("--wrap", action="store_true", help="Enable wrap-around
adjacency checks")
    pd.add_argument("--reversible-only", action="store_true", help="Force
reversible-only mapping (exact classical correspondent)")
    pd.add_argument("--max-qubits", type=int, default=8, help="Maximum qubits
(default: 8)")
    pd.add_argument("--max-layers", type=int, default=16, help="Maximum layers
(default: 16)")
    pd.add_argument("--outdir", type=str, default="out", help="Output directory
(default: out)")
    pd.add_argument("--no-png", action="store_true", help="Skip PNG rendering
(write text outputs only)")
    pd.set_defaults(func=cmd_derive)

    # category
    pc = sub.add_parser("category", help="Accept + derive semantic category +
write outputs")
    pc.add_argument("--state", type=str, default=None, help="Accepted state as
decimal or 0b... binary")
    pc.add_argument("--in", dest="infile", type=str, default=None, help="Read
accepted state from a text file")
    pc.add_argument("--wrap", action="store_true", help="Enable wrap-around
adjacency checks")
    pc.add_argument("--reversible-only", action="store_true", help="Match derive
reversible-only semantics (x/cx only)")
    pc.add_argument("--max-qubits", type=int, default=8, help="Semantic window
width q=min(m,max_qubits) (default: 8)")
    pc.add_argument("--max-layers", type=int, default=16, help="Semantic window
height layers=min(m,max_layers) (default: 16)")
    pc.add_argument("--outdir", type=str, default="out", help="Output directory
(default: out)")
    pc.add_argument("--no-png", action="store_true", help="Skip PNG rendering
(write JSON outputs only)")
    pc.add_argument("--hide-edge-labels", action="store_true", help="Hide edge
labels on category PNG")
    pc.add_argument("--show-grid-labels", action="store_true", help="Overlay
numeric token labels on grid PNG")
    pc.add_argument("--assembly", action="store_true", help="Write a side-by-side
```

```

assembly PNG (grid + category)")
pc.set_defaults(func=cmd_category)

# menu
pm = sub.add_parser("menu", help="Interactive menu mode (fallback)")
pm.set_defaults(func=cmd_menu)

return p

def main(argv: Optional[Sequence[str]] = None) -> int:
    parser = build_parser()
    args = parser.parse_args(argv)

    # If no subcommand provided, default to menu for convenience
    if not getattr(args, "cmd", None):
        return cmd_menu(args)

    # Require one of --state / --in for accept/derive/category
    if args.cmd in ("accept", "derive", "category") and not args.state and not
args.infile:
        parser.error("accept/derive/category require --state or --in")

    return int(args.func(args))

if __name__ == "__main__":
    raise SystemExit(main())

```

## [66] Hardware/out\_cat4/category.json

- **Bytes:** 10646
- **Type:** text

```
{
  "kind": "semantic_free_category_presentation",
  "objects": [
    {
      "id": "cx(shift=4)",
      "color": "#91b472"
    },
    {
      "id": "h",
      "color": "#25d072"
    },
    {
      "id": "rx(k=21)",
      "color": "#975253"
    },
  ]
}
```

```
{  
    "id": "rx(k=5)",  
    "color": "#2bb103"  
,  
{  
    "id": "rx(k=9)",  
    "color": "#081c07"  
,  
{  
    "id": "ry(k=18)",  
    "color": "#641d90"  
,  
{  
    "id": "ry(k=22)",  
    "color": "#6189d4"  
,  
{  
    "id": "ry(k=26)",  
    "color": "#00bf78"  
,  
{  
    "id": "rz(k=19)",  
    "color": "#22bb91"  
,  
{  
    "id": "rz(k=27)",  
    "color": "#2dadd9"  
,  
{  
    "id": "s",  
    "color": "#1a4a7f"  
,  
{  
    "id": "sdg",  
    "color": "#2a0b2c"  
,  
{  
    "id": "t",  
    "color": "#727cb5"  
,  
{  
    "id": "tdg",  
    "color": "#508074"  
,  
{  
    "id": "y",  
    "color": "#409378"  
}  
],  
"generators": [  
{  
    "src": "cx(shift=4)",  
    "dst": "tdg",  

```

```
"count": 1,
"label": "E\u00d71",
"examples": [
  {
    "at": [
      3,
      3
    ]
  }
],
{
  "src": "cx(shift=4)",
  "dst": "tdg",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        3,
        3
      ]
    }
  ]
},
{
  "src": "h",
  "dst": "rz(k=19)",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        1,
        4
      ]
    }
  ]
},
{
  "src": "rx(k=21)",
  "dst": "cx(shift=4)",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        2,
        3
      ]
    }
  ]
}
```

```
        ]
    },
{
  "src": "rx(k=21)",
  "dst": "rz(k=19)",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        2,
        3
      ]
    }
  ],
  "src": "rx(k=5)",
  "dst": "t",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        1,
        2
      ]
    }
  ],
  "src": "rx(k=5)",
  "dst": "tdg",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        1,
        2
      ]
    }
  ],
  "src": "rx(k=9)",
  "dst": "rx(k=5)",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
```

```
{  
    "at": [  
        0,  
        2  
    ]  
},  
{  
    "src": "rx(k=9)",  
    "dst": "rz(k=27)",  
    "kind": "E",  
    "count": 1,  
    "label": "E\u00d71",  
    "examples": [  
        {  
            "at": [  
                0,  
                2  
            ]  
        }  
    ],  
},  
{  
    "src": "ry(k=18)",  
    "dst": "rx(k=5)",  
    "kind": "E",  
    "count": 1,  
    "label": "E\u00d71",  
    "examples": [  
        {  
            "at": [  
                1,  
                1  
            ]  
        }  
    ],  
},  
{  
    "src": "ry(k=18)",  
    "dst": "y",  
    "kind": "S",  
    "count": 1,  
    "label": "S\u00d71",  
    "examples": [  
        {  
            "at": [  
                1,  
                1  
            ]  
        }  
    ],  
},  
{
```



```
        4
    ]
}
],
{
  "src": "rz(k=27)",
  "dst": "s",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        2,
        0
      ]
    }
  ],
  "src": "rz(k=27)",
  "dst": "sdg",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        0,
        3
      ]
    }
  ],
  "src": "rz(k=27)",
  "dst": "tdg",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        0,
        3
      ]
    }
  ],
  {
    "src": "rz(k=27)",
    "dst": "y",
    "kind": "E",
```

```
"count": 1,
"label": "E\u00d71",
"examples": [
  {
    "at": [
      2,
      0
    ]
  }
],
{
  "src": "s",
  "dst": "ry(k=18)",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        1,
        0
      ]
    }
  ]
},
{
  "src": "s",
  "dst": "ry(k=22)",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        3,
        0
      ]
    }
  ]
},
{
  "src": "s",
  "dst": "rz(k=27)",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        1,
        0
      ]
    }
  ]
}
```

```
        ]
    },
{
  "src": "s",
  "dst": "y",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        3,
        0
      ]
    }
  ],
  "src": "sdg",
  "dst": "h",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        0,
        4
      ]
    }
  ],
  "src": "t",
  "dst": "rx(k=21)",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        2,
        2
      ]
    }
  ],
  "src": "t",
  "dst": "ry(k=26)",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
```

```
{  
    "at": [  
        4,  
        1  
    ]  
},  
{  
    "src": "t",  
    "dst": "y",  
    "kind": "S",  
    "count": 1,  
    "label": "S\u00d71",  
    "examples": [  
        {  
            "at": [  
                2,  
                2  
            ]  
        }  
    ]  
},  
{  
    "src": "tdg",  
    "dst": "h",  
    "kind": "E",  
    "count": 1,  
    "label": "E\u00d71",  
    "examples": [  
        {  
            "at": [  
                1,  
                3  
            ]  
        }  
    ]  
},  
{  
    "src": "tdg",  
    "dst": "rx(k=21)",  
    "kind": "S",  
    "count": 1,  
    "label": "S\u00d71",  
    "examples": [  
        {  
            "at": [  
                1,  
                3  
            ]  
        }  
    ]  
},  
{
```

```
"src": "tdg",
"dst": "rx(k=9)",
"kind": "E",
"count": 1,
"label": "E\u00d71",
"examples": [
  {
    "at": [
      0,
      1
    ]
  }
],
{
  "src": "tdg",
  "dst": "ry(k=18)",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        4,
        3
      ]
    }
  ]
},
{
  "src": "tdg",
  "dst": "ry(k=18)",
  "kind": "S",
  "count": 2,
  "label": "S\u00d72",
  "examples": [
    {
      "at": [
        0,
        1
      ]
    },
    {
      "at": [
        3,
        4
      ]
    }
  ]
},
{
  "src": "tdg",
  "dst": "s",
  "kind": "S",
```

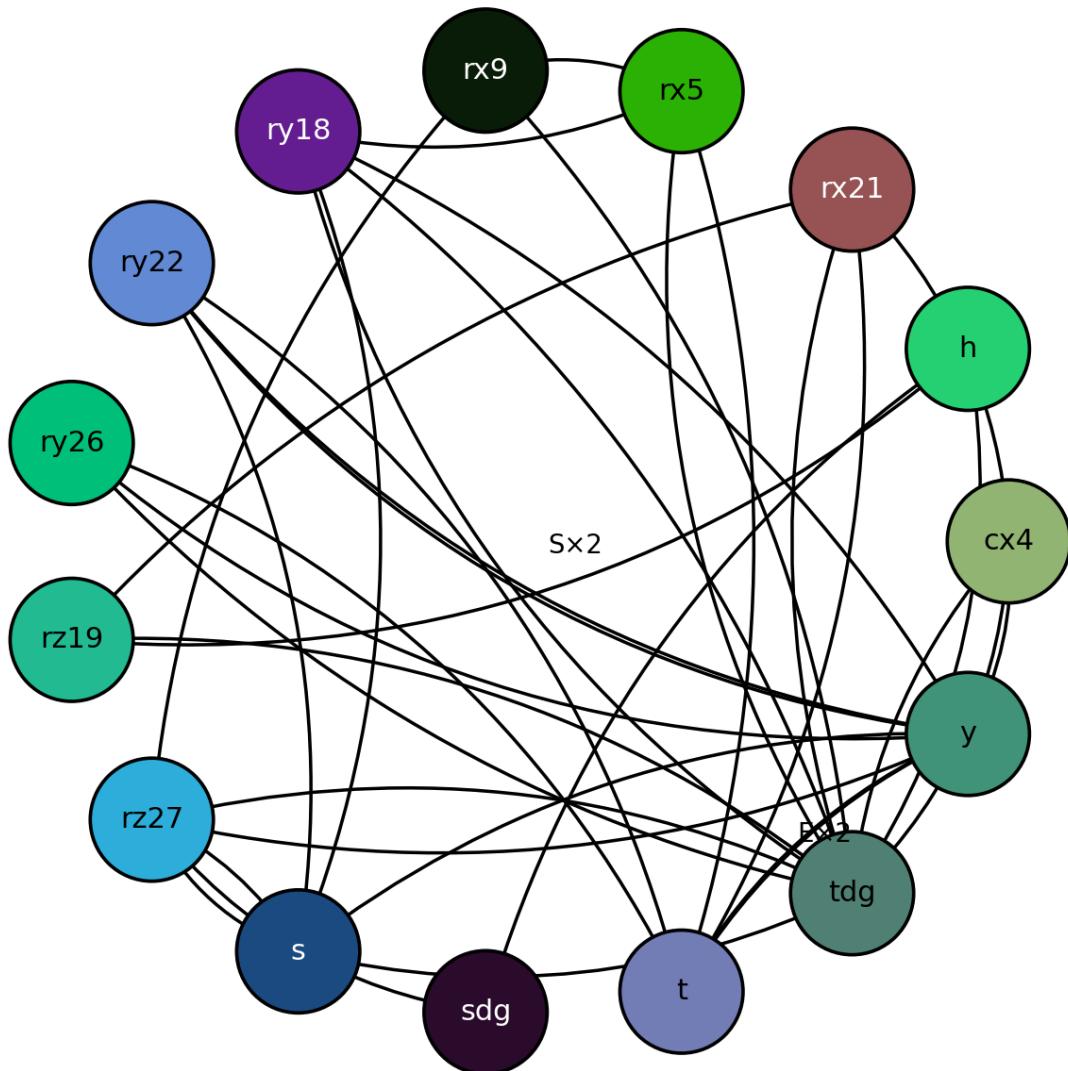
```
"count": 1,
"label": "S\u00d71",
"examples": [
  {
    "at": [
      0,
      0
    ]
  }
],
{
  "src": "tdg",
  "dst": "tdg",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        0,
        0
      ]
    }
  ]
},
{
  "src": "y",
  "dst": "cx(shift=4)",
  "kind": "E",
  "count": 1,
  "label": "E\u00d71",
  "examples": [
    {
      "at": [
        3,
        2
      ]
    }
  ]
},
{
  "src": "y",
  "dst": "ry(k=22)",
  "kind": "S",
  "count": 1,
  "label": "S\u00d71",
  "examples": [
    {
      "at": [
        2,
        1
      ]
    }
  ]
}
```

```
        ]
    },
    {
        "src": "y",
        "dst": "ry(k=26)",
        "kind": "S",
        "count": 1,
        "label": "S\u00d71",
        "examples": [
            {
                "at": [
                    3,
                    2
                ]
            }
        ],
        {
            "src": "y",
            "dst": "t",
            "kind": "E",
            "count": 2,
            "label": "E\u00d72",
            "examples": [
                {
                    "at": [
                        2,
                        1
                    ]
                },
                {
                    "at": [
                        4,
                        0
                    ]
                }
            ]
        }
    ],
    "composition": "paths (free category on the generator graph)",
    "identities": "implicit for each object",
    "window": {
        "q": 5,
        "layers": 5,
        "m": 5
    },
    "reversible_only": false
}
```

[67] Hardware/out\_cat4/category.png

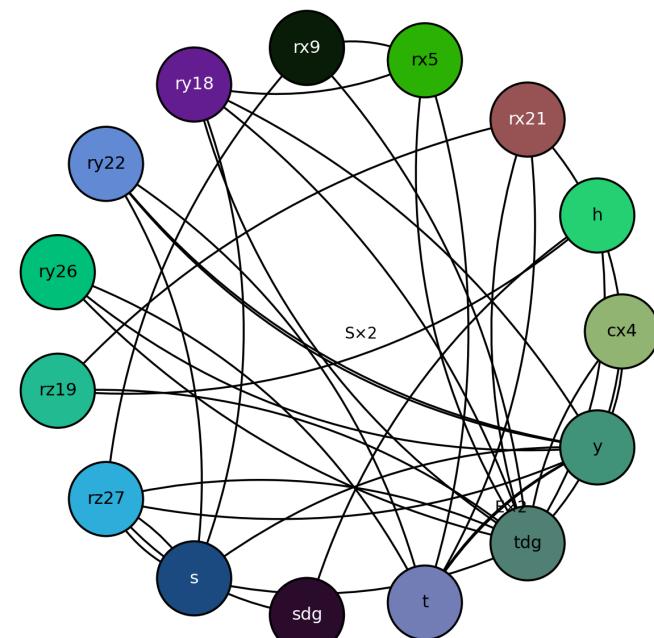
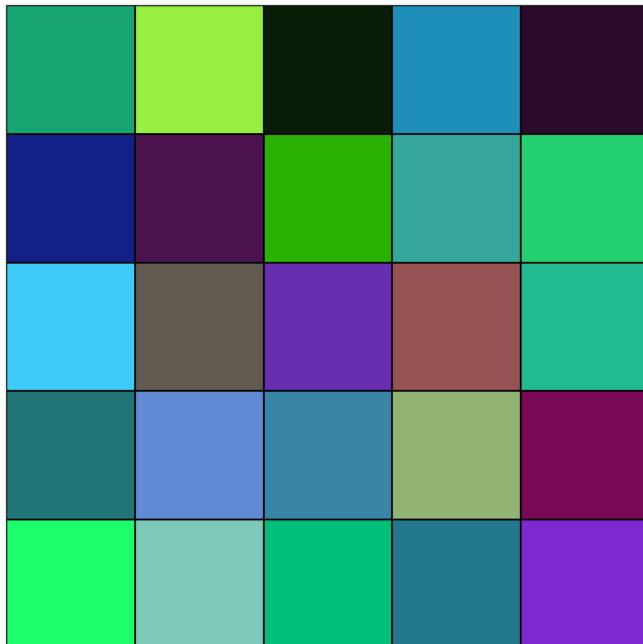
- **Bytes:** 289923

- **Type:** png
- **Dimensions:** 1443x1443
- **Path (from doc):** ../Hardware/out\_cat4/category.png



[68] Hardware/out\_cat4/category\_assembly.png

- **Bytes:** 338127
- **Type:** png
- **Dimensions:** 2397x1213
- **Path (from doc):** ../Hardware/out\_cat4/category\_assembly.png



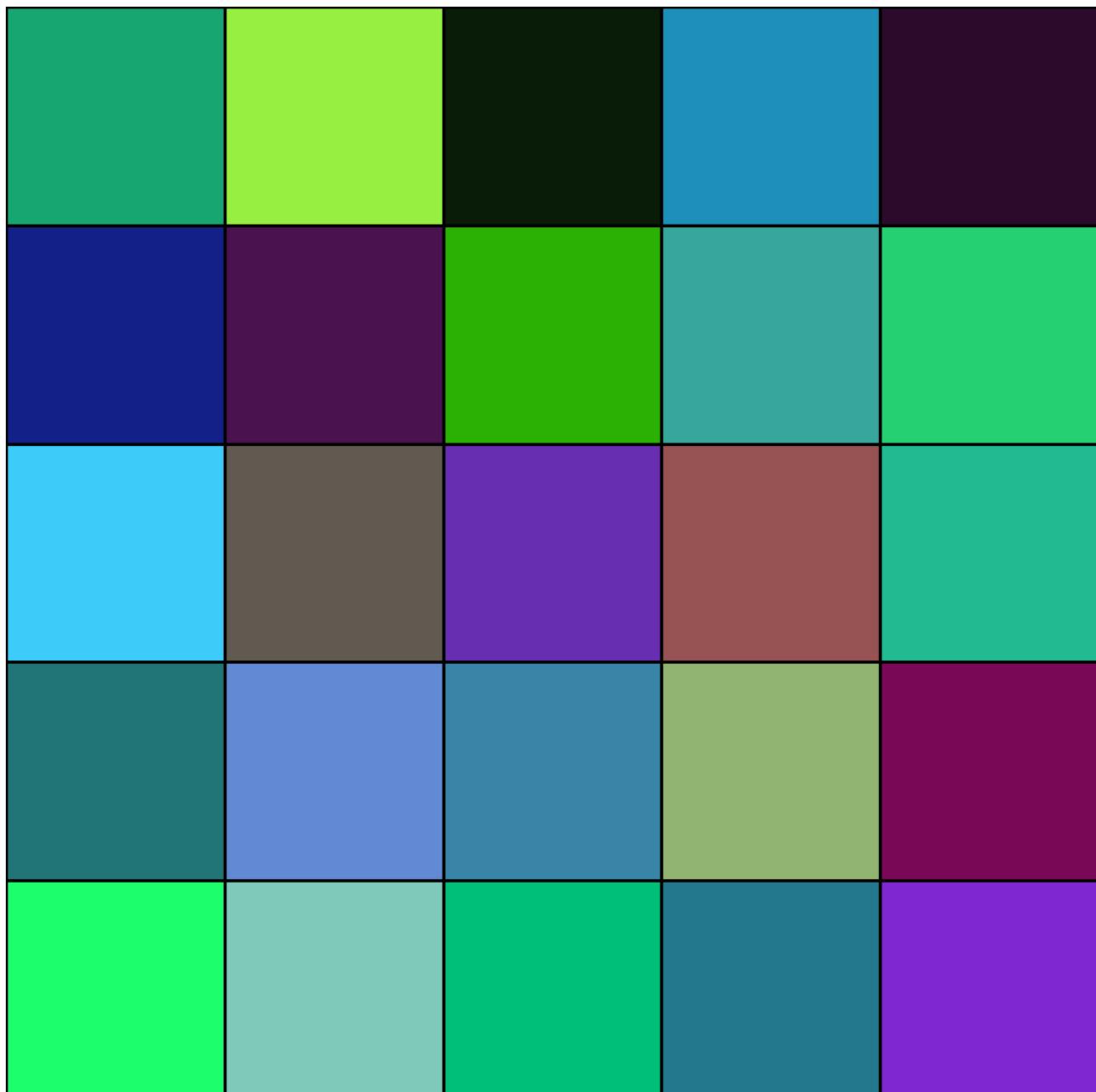
## [69] [Hardware/out\\_cat4/category\\_manifest.json](#)

- **Bytes:** 370
- **Type:** text

```
{
  "accepted": true,
  "grid_m": 5,
  "token_count": 25,
  "wrap_adjacency": false,
  "reversible_only": false,
  "max_qubits": 8,
  "max_layers": 16,
  "outputs": [
    "out_cat4\\category.json",
    "out_cat4\\grid.png",
    "out_cat4\\category.png",
    "out_cat4\\category_assembly.png"
  ],
  "category_kind": "semantic_free_category_presentation"
}
```

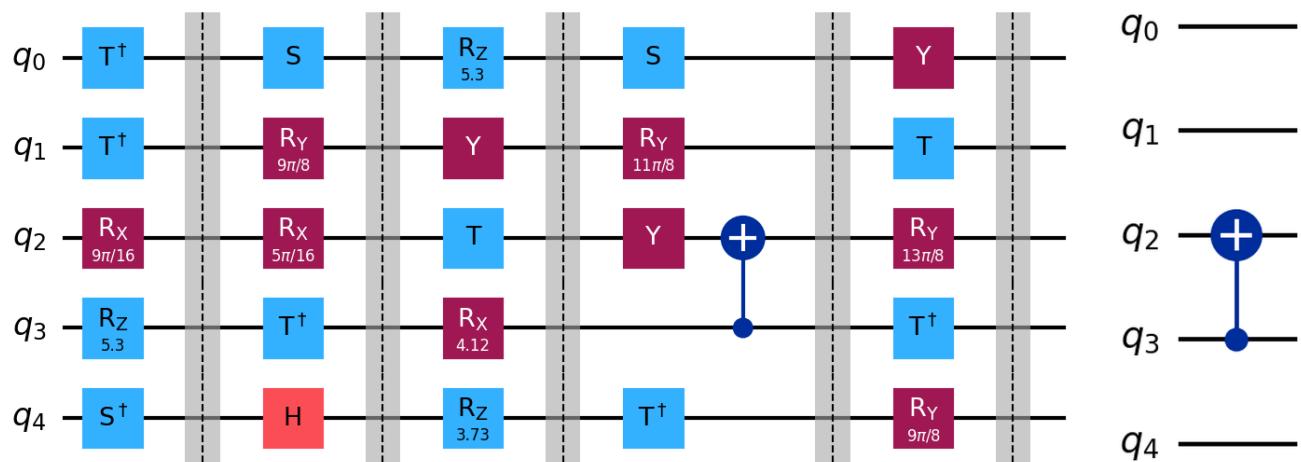
## [70] [Hardware/out\\_cat4/grid.png](#)

- **Bytes:** 9256
- **Type:** png
- **Dimensions:** 1155×1155
- **Path (from doc):** ../Hardware/out\_cat4/grid.png



[71] [Hardware/outnn\\_4\\_30-17/assembly.png](#)

- **Bytes:** 51217
- **Type:** png
- **Dimensions:** 1461×563
- **Path (from doc):** ../Hardware/outnn\_4\_30-17/assembly.png



[72] [Hardware/outnn\\_4\\_30-17/classical.png](#)

- **Bytes:** 9377
- **Type:** png
- **Dimensions:** 265×551
- **Path (from doc):** [..../Hardware/outnn\\_4\\_30-17/classical.png](#)

$q_0$  —

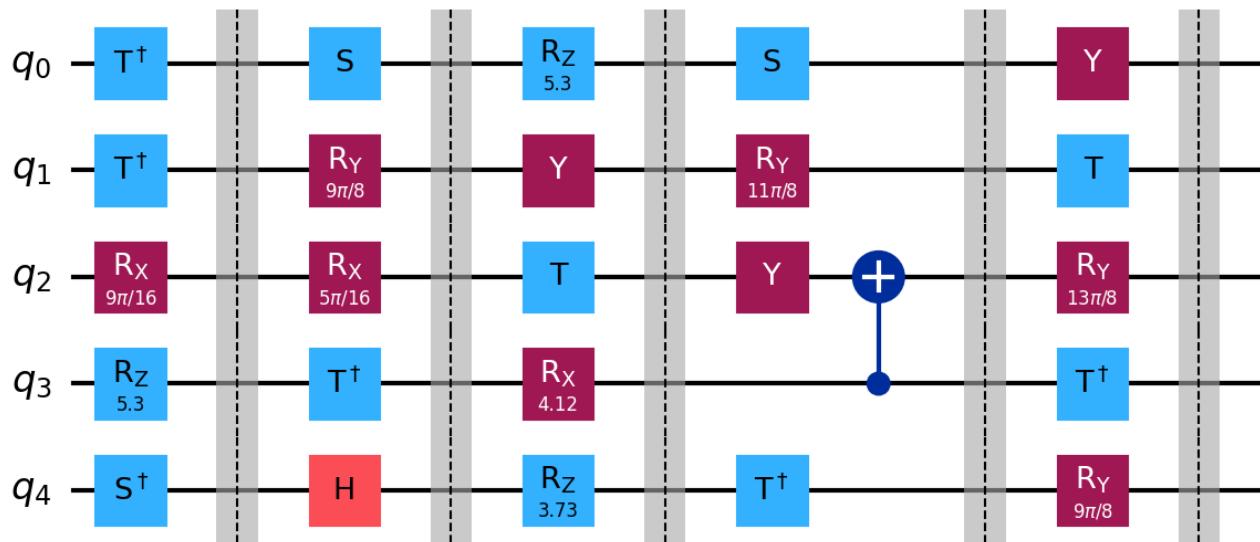
$q_1$  —

$q_2$  —  
+  
 $q_3$  —

$q_4$  —

[73] [Hardware/outnn\\_4\\_30-17/quantum.png](#)

- **Bytes:** 36245
- **Type:** png
- **Dimensions:** 1231×551
- **Path (from doc):** ../Hardware/outnn\_4\_30-17/quantum.png



[74] Hardware/outnn\_4\_30-17/source\_classical.txt

- **Bytes:** 9
- **Type:** text

```
cx(3,2)
```

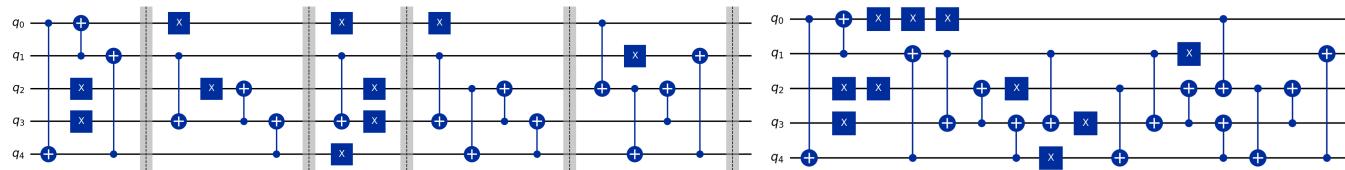
[75] Hardware/outnn\_4\_30-17/source\_quantum.txt

- **Bytes:** 335
- **Type:** text

```
tdg(0) tdg(1) rx(1.7671458676442586,2) rz(5.301437602932776,3) sdg(4) s(0)
ry(3.5342917352885173,1) rx(0.9817477042468103,2) tdg(3) h(4)
rz(5.301437602932776,0) y(1) t(2) rx(4.123340357836604,3) rz(3.730641276137879,4)
s(0) ry(4.319689898685965,1) y(2) cx(3,2) tdg(4) y(0) t(1) ry(5.105088062083414,2)
tdg(3) ry(3.5342917352885173,4)
```

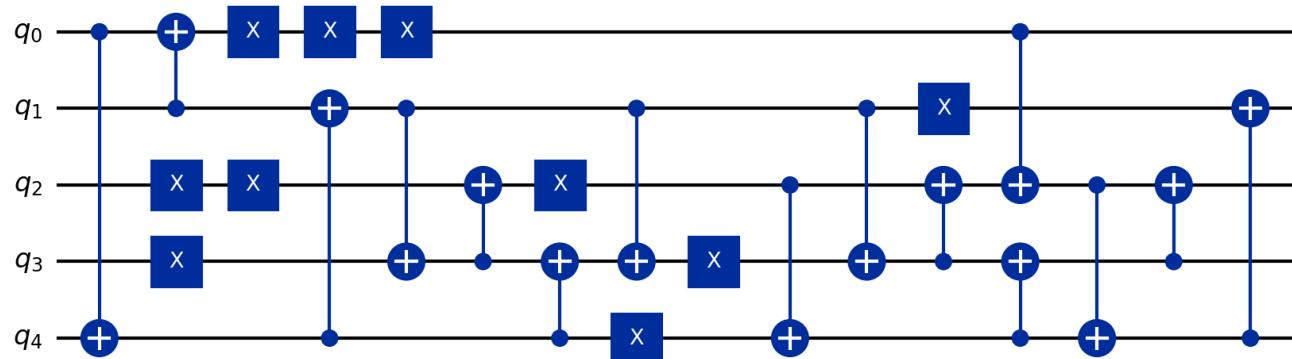
[76] Hardware/outny\_4\_30-17/assembly.png

- **Bytes:** 140434
- **Type:** png
- **Dimensions:** 4038×563
- **Path (from doc):** ../Hardware/outny\_4\_30-17/assembly.png



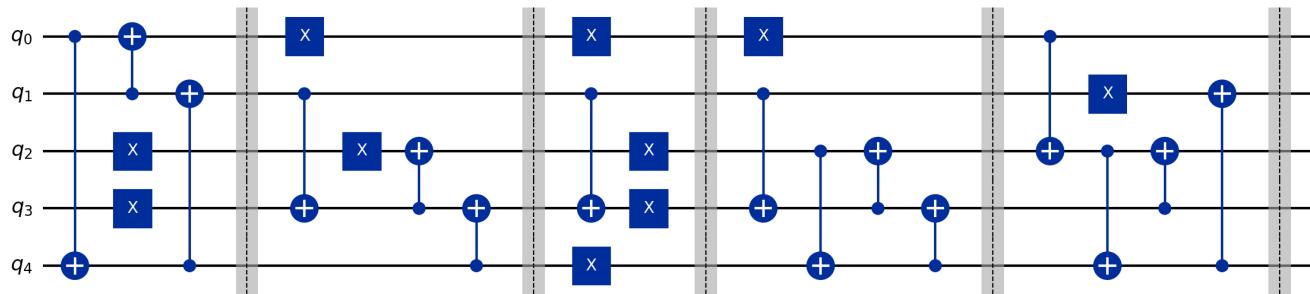
[77] Hardware/outny\_4\_30-17/classical.png

- **Bytes:** 42350
- **Type:** png
- **Dimensions:** 1714×551
- **Path (from doc):** ../Hardware/outny\_4\_30-17/classical.png



[78] Hardware/outny\_4\_30-17/quantum.png

- **Bytes:** 49478
- **Type:** png
- **Dimensions:** 2293×551
- **Path (from doc):** ../Hardware/outny\_4\_30-17/quantum.png



[79] Hardware/outny\_4\_30-17/source\_classical.txt

- **Bytes:** 171
- **Type:** text

```
cx(0,4) cx(1,0) x(2) x(3) cx(4,1) x(0) cx(1,3) x(2) cx(3,2) cx(4,3) x(0) cx(1,3)
x(2) x(3) x(4) x(0) cx(1,3) cx(2,4) cx(3,2) cx(4,3) cx(0,2) x(1) cx(2,4) cx(3,2)
cx(4,1)
```

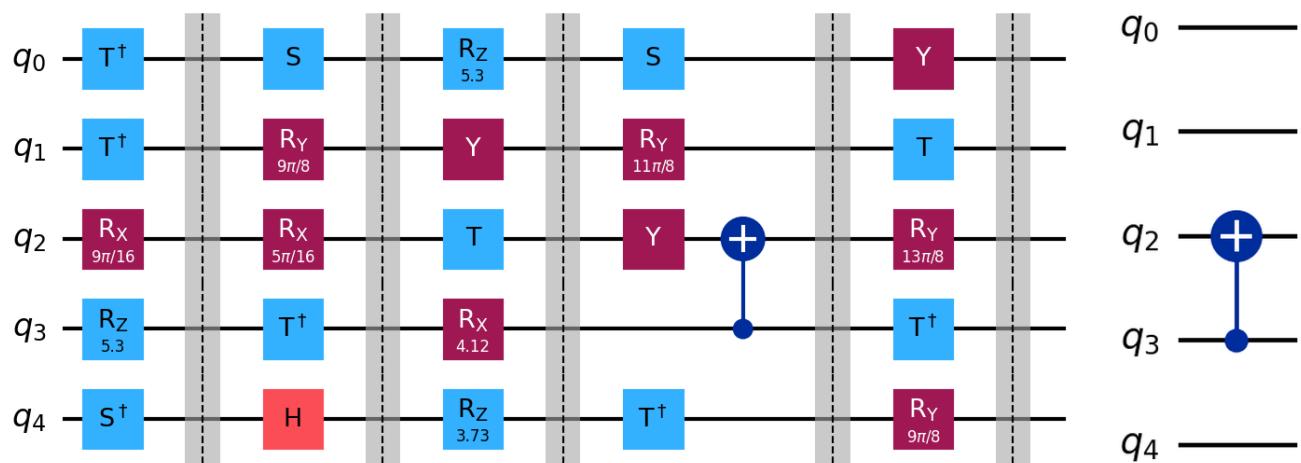
## [80] Hardware/outny\_4\_30-17/source\_quantum.txt

- **Bytes:** 171
- **Type:** text

```
cx(0,4) cx(1,0) x(2) x(3) cx(4,1) x(0) cx(1,3) x(2) cx(3,2) cx(4,3) x(0) cx(1,3)
x(2) x(3) x(4) x(0) cx(1,3) cx(2,4) cx(3,2) cx(4,3) cx(0,2) x(1) cx(2,4) cx(3,2)
cx(4,1)
```

## [81] Hardware/outyn\_4\_30-17/assembly.png

- **Bytes:** 51217
- **Type:** png
- **Dimensions:** 1461×563
- **Path (from doc):** ../Hardware/outyn\_4\_30-17/assembly.png



## [82] Hardware/outyn\_4\_30-17/classical.png

- **Bytes:** 9377
- **Type:** png
- **Dimensions:** 265×551
- **Path (from doc):** ../Hardware/outyn\_4\_30-17/classical.png

$q_0$  —————

$q_1$  —————

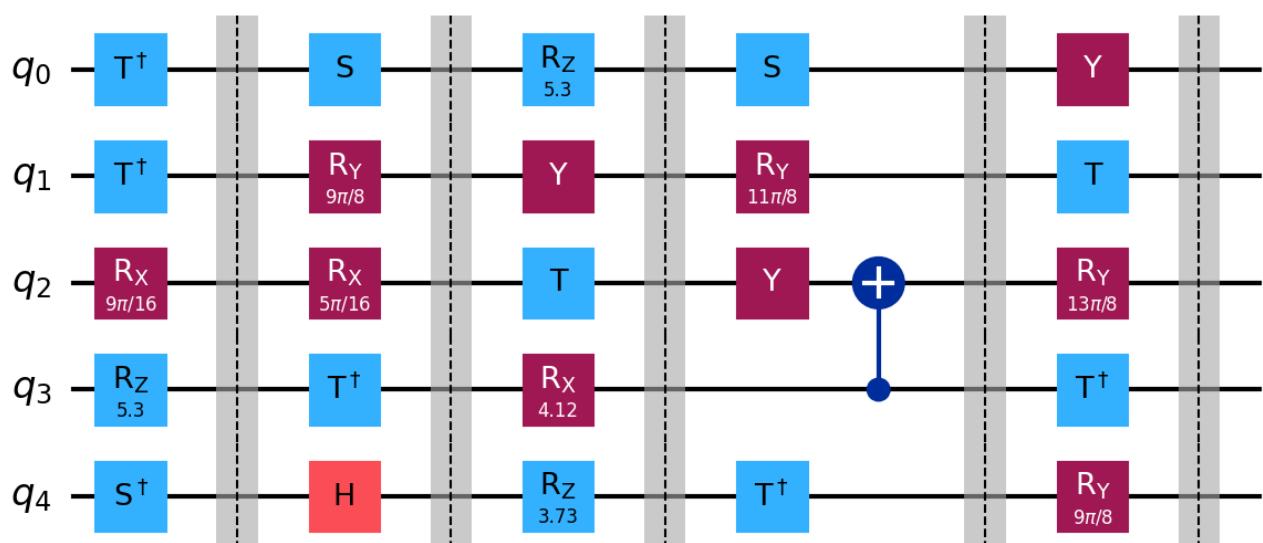
$q_2$  ————— +

$q_3$  —————

$q_4$  —————

[83] [Hardware/outyn\\_4\\_30-17/quantum.png](#)

- **Bytes:** 36245
- **Type:** png
- **Dimensions:** 1231×551
- **Path (from doc):** ../../Hardware/outyn\_4\_30-17/quantum.png



## [84] Hardware/outyn\_4\_30-17/source\_classical.txt

- **Bytes:** 9
- **Type:** text

```
cx(3,2)
```

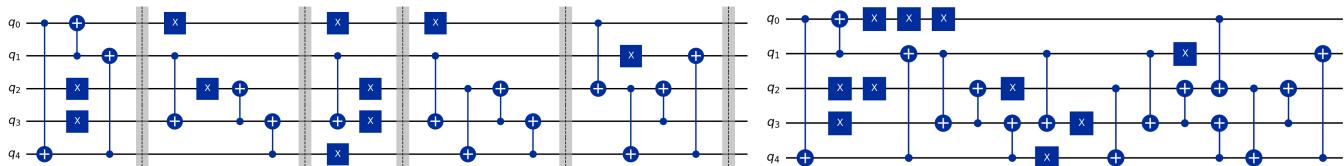
## [85] Hardware/outyn\_4\_30-17/source\_quantum.txt

- **Bytes:** 335
- **Type:** text

```
tdg(0) tdg(1) rx(1.7671458676442586,2) rz(5.301437602932776,3) sdg(4) s(0)
ry(3.5342917352885173,1) rx(0.9817477042468103,2) tdg(3) h(4)
rz(5.301437602932776,0) y(1) t(2) rx(4.123340357836604,3) rz(3.730641276137879,4)
s(0) ry(4.319689898685965,1) y(2) cx(3,2) tdg(4) y(0) t(1) ry(5.105088062083414,2)
tdg(3) ry(3.5342917352885173,4)
```

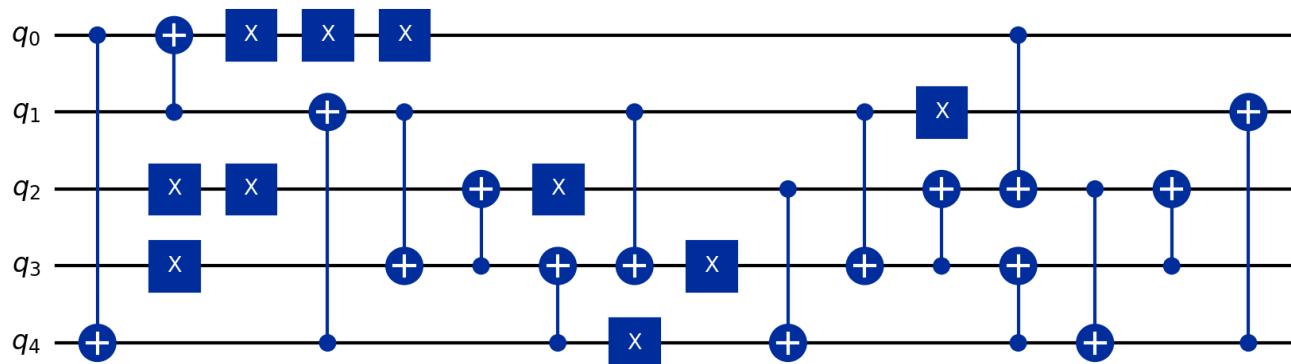
## [86] Hardware/outyy\_4\_30-17/assembly.png

- **Bytes:** 140434
- **Type:** png
- **Dimensions:** 4038×563
- **Path (from doc):** ../Hardware/outyy\_4\_30-17/assembly.png



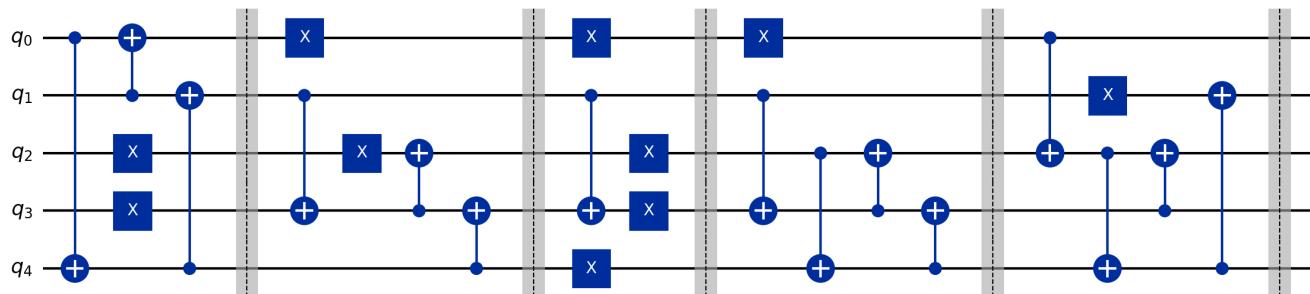
## [87] Hardware/outyy\_4\_30-17/classical.png

- **Bytes:** 42350
- **Type:** png
- **Dimensions:** 1714×551
- **Path (from doc):** ../Hardware/outyy\_4\_30-17/classical.png



[88] [Hardware/outyy\\_4\\_30-17/quantum.png](#)

- **Bytes:** 49478
- **Type:** png
- **Dimensions:** 2293×551
- **Path (from doc):** ../Hardware/outyy\_4\_30-17/quantum.png



[89] [Hardware/outyy\\_4\\_30-17/source\\_classical.txt](#)

- **Bytes:** 171
- **Type:** text

```
cx(0,4) cx(1,0) x(2) x(3) cx(4,1) x(0) cx(1,3) x(2) cx(3,2) cx(4,3) x(0) cx(1,3)
x(2) x(3) x(4) x(0) cx(1,3) cx(2,4) cx(3,2) cx(4,3) cx(0,2) x(1) cx(2,4) cx(3,2)
cx(4,1)
```

[90] [Hardware/outyy\\_4\\_30-17/source\\_quantum.txt](#)

- **Bytes:** 171
- **Type:** text

```
cx(0,4) cx(1,0) x(2) x(3) cx(4,1) x(0) cx(1,3) x(2) cx(3,2) cx(4,3) x(0) cx(1,3)
x(2) x(3) x(4) x(0) cx(1,3) cx(2,4) cx(3,2) cx(4,3) cx(0,2) x(1) cx(2,4) cx(3,2)
cx(4,1)
```

## [91] integerDatabase/operator-alphabet.js

- **Bytes:** 42865
- **Type:** text

```
/**  
 * operator-alphabet.js  
 *  
 * This module implements the Alphabet System (Tile Color Viewer) for the Operator  
 framework,  
 * providing grid visualization and manipulation capabilities.  
 * It integrates with the operator-dropdown.js system.  
 *  
 * @module OperatorAlphabet  
 * @version 1.0.0  
 * @depends OperatorFramework.Dropdown  
 */  
  
// Access the shared namespace for Operator framework  
  
/**  
 * Alphabet System for grid visualization and tile color operations  
 * @namespace OperatorAlphabet  
 */  
OperatorFramework.Alphabet = (function() {  
    'use strict';  
  
    // Private module variables  
    let _initialized = false;  
    let _section = null;  
    let _grids = []; // Array to store color sequences of each grid  
    let _currentGridIndex = 0; // Index of the currently displayed grid  
    let _defaultOptions = {  
        containerId: 'main', // ID of the container to add the alphabet section to  
        sectionId: 'alphabet-system-section', // ID for the created section  
        element:  
            defaultTileColor: '#ff0000', // Default tile color  
            defaultTileSize: 100 // Default tile size in pixels  
    };  
    let _options = {};  
  
    /**  
     * Initialize the Alphabet System  
     * @param {Object} options - Configuration options  
     * @param {string} [options.containerId='main'] - ID of the container to add  
     * the alphabet section to  
     * @param {string} [options.sectionId='alphabet-system-section'] - ID for the  
     * created section element  
     * @param {string} [options.defaultTileColor='#ff0000'] - Default tile color  
     * @param {number} [options.defaultTileSize=100] - Default tile size in pixels  
     * @returns {boolean} Success status  
    */
```

```
function initialize(options = {}) {
    if (_initialized) {
        console.warn('OperatorAlphabet is already initialized');
        return false;
    }

    // Merge options with defaults
    _options = { ..._defaultOptions, ...options };

    console.log('OperatorAlphabet initialized with default options');
    _initialized = true;

    // Register with dropdown system if available
    if (OperatorFramework.Dropdown) {
        OperatorFramework.Dropdown.addSystem({
            id: 'alphabet',
            name: 'Tile Color Viewer',
            description: 'Create and manipulate colorful grid visualizations',
            loadFunction: loadAlphabetSystem
        });
        console.log('Alphabet System registered with
OperatorFramework.Dropdown');
    } else {
        console.warn('OperatorFramework.Dropdown not available, Alphabet
System not registered');
    }
}

return true;
}

/**
 * Load the Alphabet System UI and functionality
 * @returns {boolean} Success status
 */
function loadAlphabetSystem() {
    if (!initialized) {
        console.error('OperatorAlphabet not initialized, call initialize()
first');
        return false;
    }

    // Check if already loaded
    if (_section && document.getElementById(_options.sectionId)) {
        showAlphabetSection();
        return true;
    }

    // Get container
    //const container = document.getElementById(_options.containerId);
    // Use:
    const container = document.querySelector('main');
    if (!container) {
        console.error(`Container with ID "${_options.containerId}" not
found`);
    }
}
```

```
        return false;
    }

    // Create Alphabet System section
    _section = document.createElement('section');
    _section.id = _options.sectionId;
    _section.className = 'operation-section';

    // Populate with Alphabet System content
    _section.innerHTML = _getAlphabetSystemHTML();

    // Add to container
    container.appendChild(_section);

    // Add styles
    _addAlphabetSystemStyles();

    // Initialize event listeners
    _initAlphabetSystemEventListeners();

    // Show the section
    showAlphabetSection();

    // Initialize the grid with default settings
    updateGrid();

    console.log('Alphabet System loaded successfully');
    return true;
}

/**
 * Show the Alphabet System section and hide other sections
 */
function showAlphabetSection() {
    if (!_section) return;

    // Hide all sections
    const allSections = document.querySelectorAll('.operation-section');
    allSections.forEach(section => {
        section.classList.remove('active');
    });

    // Show Alphabet section
    _section.classList.add('active');
}

/**
 * Generate the Alphabet System HTML content
 * @private
 * @returns {string} HTML content
 */
function _getAlphabetSystemHTML() {
    return `
        <h2>Tile Color Viewer</h2>
        <div>
            <table border="1">
                <thead>
                    <tr>
                        <th>Color Name</th>
                        <th>Hex Code</th>
                        <th>RGB Values</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Red</td>
                        <td>#FF0000</td>
                        <td>255, 0, 0</td>
                    </tr>
                    <tr>
                        <td>Green</td>
                        <td>#00FF00</td>
                        <td>0, 255, 0</td>
                    </tr>
                    <tr>
                        <td>Blue</td>
                        <td>#0000FF</td>
                        <td>0, 0, 255</td>
                    </tr>
                    <tr>
                        <td>Yellow</td>
                        <td>#FFFF00</td>
                        <td>255, 255, 0</td>
                    </tr>
                    <tr>
                        <td>Magenta</td>
                        <td>#FF00FF</td>
                        <td>255, 0, 255</td>
                    </tr>
                    <tr>
                        <td>Cyan</td>
                        <td>#00FFFF</td>
                        <td>0, 255, 255</td>
                    </tr>
                    <tr>
                        <td>Black</td>
                        <td>#000000</td>
                        <td>0, 0, 0</td>
                    </tr>
                    <tr>
                        <td>White</td>
                        <td>#FFFFFF</td>
                        <td>255, 255, 255</td>
                    </tr>
                </tbody>
            </table>
        </div>
    `;
}
```

```
<div class="operation-container">
    <div class="operation-controls">
        <!-- Hero Section -->
        <div class="alphabet-hero">
            <p>Create and customize colorful grids with ease.</p>
        </div>

        <!-- Controls Section -->
        <div id="controls" class="controls-container">
            <label for="tile-color">Tile Color: </label>
            <input type="color" id="tile-color"
value="${_options.defaultTileColor}">

            <label for="tile-size">Tile Size (px): </label>
            <input type="number" id="tile-size"
value="${_options.defaultTileSize}" placeholder="Tile size in pixels">

            <button id="update-grid">Update Grid</button>
        </div>

        <!-- Compounded ID Input Container -->
        <div id="compounded-id-input-container" class="controls-
container">
            <h3>Generate Grids by Compounded IDs</h3>
            <div id="compounded-id-inputs">
                <div class="compounded-id-input-row">
                    <input type="text" class="compounded-id-input"
placeholder="Enter Compounded Grid ID">
                </div>
                </div>
                <button id="add-compounded-id">+</button>
                <button id="generate-grid-by-id">Generate Grids</button>
                <br>
                <!-- File Upload Input -->
                <p>Upload a file containing compounded grid IDs (e.g.,
sequences.txt).</p>
                <input type="file" id="file-input" accept=".txt">

                <!-- Execute Button -->
                <button id="execute-button" aria-label="Process uploaded
file">Execute</button>
            </div>

            <!-- Color List Container -->
            <div id="color-list-container" class="controls-container">
                <h3>Sequence List</h3>
                <ul id="color-list"></ul>
                <button id="download-colors">Download Tile ID</button>
            </div>
        </div>

        <div class="operation-result">
            <h3>Grid Display</h3>
```

```
<!-- Grid Display Section -->
<canvas id="grid-canvas" style="border: 1px solid #ccc;">
</canvas>

<!-- Navigation Controls -->
<div id="navigation-controls">
    <button id="prev-grid" aria-label="Go to previous
grid">Previous Grid</button>
    <button id="next-grid" aria-label="Go to next grid">Next
Grid</button>
    <button id="download-grid-image" aria-label="Download
current grid as image">Download Grid Image</button>
</div>

<!-- Current Grid Information -->
<div id="current-grid-id"></div>
<div id="current-grid-position"></div>
</div>
</div>
`;

}

/** 
 * Add CSS styles for the Alphabet System
 * @private
 */
function _addAlphabetSystemStyles() {
    const styleId = 'operator-alphabet-styles';

    // Don't add styles if they already exist
    if (document.getElementById(styleId)) return;

    const styleElement = document.createElement('style');
    styleElement.id = styleId;
    styleElement.textContent = `
        /* CSS for Alphabet System */
        #alphabet-system-section {
            --primary-color: #4B5320; /* Army Green */
            --secondary-color: #FFFFFF; /* White */
            --accent-color: #BDB76B; /* Dark Khaki */
            --accent-hover-color: #D2B48C; /* Tan */
            --background-color: #F5F5DC; /* Beige */
            --text-color: #333333; /* Dark Gray */
        }

        /* General Reset - scoped to alphabet section */
        #alphabet-system-section * {
            box-sizing: border-box;
        }

        .alphabet-hero {
            width: 100%;
            text-align: center;
            padding: 20px;
        }
    `;
}
```

```
background-color: var(--primary-color);
color: var(--secondary-color);
margin-bottom: 20px;
}

.alphabet-hero p {
    font-size: 1.2rem;
}

/* Controls Container */
.controls-container {
    background: var(--secondary-color);
    padding: 20px;
    border: 2px solid var(--accent-color);
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.4);
    width: 100%;
    margin-bottom: 20px;
}

#controls {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    align-items: center;
    gap: 10px;
}

#controls label {
    flex: 1 1 200px;
    font-size: 1rem;
    color: var(--primary-color);
    font-weight: 500;
}

#controls input {
    flex: 1 1 200px;
    padding: 8px 12px;
    border: 2px solid var(--accent-color);
    font-size: 1rem;
    color: var(--text-color);
    background-color: #F5F5DC; /* Beige */
    font-family: "Fira Code", "Consolas", monospace;
}

#controls button {
    flex: 1 1 100%;
    padding: 12px 25px;
    font-size: 1rem;
    color: var(--secondary-color);
    background-color: var(--primary-color);
    border: 2px solid var(--accent-color);
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease, color
0.3s ease;
```

```
    margin-top: 10px;
    font-family: "Fira Code", "Consolas", monospace;
}

#controls button:hover {
    background-color: var(--accent-color);
    color: var(--text-color);
    transform: translateY(-2px);
}

/* Grid Canvas */
#grid-canvas {
    border: 2px solid var(--accent-color);
    margin-bottom: 20px;
    width: 100%;
    max-width: 400px;
    height: auto;
}

/* Navigation Controls */
#navigation-controls {
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 10px;
    width: 100%;
    margin-bottom: 20px;
    flex-wrap: wrap;
}

#navigation-controls button {
    padding: 12px 25px;
    font-size: 1rem;
    color: var(--secondary-color);
    background-color: var(--primary-color);
    border: 2px solid var(--accent-color);
    cursor: pointer;
    font-weight: bold;
    transition: background-color 0.3s ease, transform 0.2s ease, color
0.3s ease;
    font-family: "Fira Code", "Consolas", monospace;
}

#navigation-controls button:hover {
    background-color: var(--accent-color);
    color: var(--text-color);
    transform: translateY(-2px);
}

#navigation-controls button:disabled {
    background-color: #ccc;
    border: 2px solid #999;
    color: #666;
    cursor: not-allowed;
```

```
        transform: none;
    }

/* Current Grid Information */
#current-grid-id,
#current-grid-position {
    font-size: 1rem;
    color: var(--primary-color);
    margin-bottom: 10px;
    text-align: center;
    font-weight: 500;
    font-family: "Fira Code", "Consolas", monospace;
}

/* Color List Container */
#color-list-container h3 {
    font-size: 1.2rem;
    margin-bottom: 10px;
    color: var(--primary-color);
    font-weight: 600;
    text-transform: uppercase;
    letter-spacing: 1px;
}

#color-list {
    list-style-type: none;
    padding: 0;
    margin: 0;
}

#color-list li {
    background: #F5F5DC; /* Beige */
    padding: 10px;
    margin-bottom: 5px;
    border: 2px solid var(--accent-color);
    font-size: 1rem;
    color: var(--text-color);
    font-family: "Fira Code", "Consolas", monospace;
}

/* Download Colors Button */
#download-colors {
    margin-top: 10px;
    padding: 12px 25px;
    font-size: 1rem;
    color: var(--secondary-color);
    background-color: var(--primary-color);
    border: 2px solid var(--accent-color);
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease, color
0.3s ease;
    font-family: "Fira Code", "Consolas", monospace;
}
```

```
#download-colors:hover {
    background-color: var(--accent-color);
    color: var(--text-color);
    transform: translateY(-2px);
}

/* Compounded ID Input Container */
#compounded-id-inputs {
    display: flex;
    flex-direction: column;
    gap: 10px;
    margin-bottom: 10px;
}

.compounded-id-input-row {
    display: flex;
    align-items: center;
}

.compounded-id-input {
    flex: 1;
    padding: 8px 12px;
    border: 2px solid var(--accent-color);
    font-size: 1rem;
    color: var(--text-color);
    background-color: #F5F5DC; /* Beige */
    font-family: "Fira Code", "Consolas", monospace;
}

/* Add Compounded ID Button */
#add-compounded-id {
    padding: 10px;
    font-size: 1.5rem;
    color: var(--secondary-color);
    background-color: var(--primary-color);
    border: 2px solid var(--accent-color);
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease, color
0.3s ease;
    margin-bottom: 10px;
    width: 40px;
    height: 40px;
    line-height: 20px;
    font-family: "Fira Code", "Consolas", monospace;
}

#add-compounded-id:hover {
    background-color: var(--accent-color);
    color: var(--text-color);
    transform: translateY(-2px);
}

/* Generate Grids Button */
#generate-grid-by-id, #execute-button {
```

```
padding: 12px 25px;
font-size: 1rem;
color: var(--secondary-color);
background-color: var(--primary-color);
border: 2px solid var(--accent-color);
cursor: pointer;
transition: background-color 0.3s ease, transform 0.2s ease, color
0.3s ease;
font-family: "Fira Code", "Consolas", monospace;
margin-top: 10px;
}

#generate-grid-by-id:hover, #execute-button:hover {
background-color: var(--accent-color);
color: var(--text-color);
transform: translateY(-2px);
}

/* Error messages */
.error-message {
color: #e74c3c;
background-color: rgba(231, 76, 60, 0.1);
padding: 10px;
border-left: 4px solid #e74c3c;
margin-top: 10px;
font-size: 0.9rem;
}

/* Success messages */
.success-message {
color: #2ecc71;
background-color: rgba(46, 204, 113, 0.1);
padding: 10px;
border-left: 4px solid #2ecc71;
margin-top: 10px;
font-size: 0.9rem;
}
};

document.head.appendChild(styleElement);
}

/**
 * Initialize event listeners for the Alphabet System
 * @private
 */
function _initAlphabetSystemEventListeners() {
// Main buttons
document.getElementById('update-grid').addEventListener('click',
updateGrid);
document.getElementById('add-compounded-id').addEventListener('click',
addNewCompoundedIDInput);
document.getElementById('generate-grid-by-id').addEventListener('click',
generateGridsFromInputs);
```

```
document.getElementById('execute-button').addEventListener('click', processUploadedFile);
document.getElementById('download-colors').addEventListener('click', downloadTileID);

// Navigation buttons
document.getElementById('prev-grid').addEventListener('click', () => {
    if (_currentGridIndex > 0) {
        _currentGridIndex--;
        displayCurrentGrid();
    }
});

document.getElementById('next-grid').addEventListener('click', () => {
    if (_currentGridIndex < _grids.length - 1) {
        _currentGridIndex++;
        displayCurrentGrid();
    }
});

document.getElementById('download-grid-image').addEventListener('click', downloadGridImage);

// Initialize with default value
const defaultCompoundedID = '1443664';
document.querySelector('.compounded-id-input').value =
defaultCompoundedID;
}

/**
 * Convert hex color code to a unique numeric ID (BigInt)
 * @param {string} hex - Hex color code
 * @returns {BigInt} Unique numeric ID
 */
function hexToID(hex) {
    return BigInt('0x' + hex.replace('#', '')) + BigInt(1);
}

/**
 * Convert numeric ID (BigInt) back to hex color code
 * @param {BigInt|string|number} id - Numeric ID
 * @returns {string} Hex color code
 */
function idToHex(id) {
    id = BigInt(id);
    if (id <= 0n) {
        return '#000000'; // Default to black
    }
    let hex = (id - BigInt(1)).toString(16).padStart(6, '0');
    return `#${hex}`;
}

/**
 * Calculate the compounded grid ID from a color sequence

```

```
* @param {Array<string>} colorSequence - Array of hex color codes
* @returns {string} Compounded grid ID
*/
function calculateCompoundedGridID(colorSequence) {
    return colorSequence.map(color => hexToID(color).toString()).join('');
}

/**
 * Calculate grid dimensions based on the number of tiles
 * @param {number} numTiles - Number of tiles
 * @returns {Object} Grid dimensions {rows, cols}
*/
function calculateGridDimensions(numTiles) {
    const sideLength = Math.ceil(Math.sqrt(numTiles));
    return { rows: sideLength, cols: sideLength };
}

/**
 * Validate hex color code
 * @param {string} hex - Hex color code
 * @returns {boolean} Whether the hex color code is valid
*/
function isValidHexColor(hex) {
    return /^[#([0-9A-F]{6})$/i.test(hex);
}

/**
 * Update the grid based on user inputs
*/
function updateGrid() {
    try {
        const tileColor = document.getElementById('tile-color').value;
        const tileSize = parseInt(document.getElementById('tile-size').value);

        if (isNaN(tileSize) || tileSize <= 0) {
            showError("Please enter a valid positive number for tile size.");
            return;
        }

        const numTiles = 1; // Start with a single tile
        const colorSequence = Array(numTiles).fill(tileColor);
        _grids = [colorSequence]; // Reset grids array with the new grid
        _currentGridIndex = 0; // Reset to the first grid

        displayCurrentGrid();

        showSuccess("Grid updated successfully");
    } catch (error) {
        showError(`Error updating grid: ${error.message}`);
        console.error("Error updating grid:", error);
    }
}

/**

```

```
* Display the current grid based on _currentGridIndex
*/
function displayCurrentGrid() {
  try {
    const canvas = document.getElementById('grid-canvas');
    const context = canvas.getContext('2d');

    if (_grids.length === 0) {
      context.clearRect(0, 0, canvas.width, canvas.height);
      return;
    }

    const tileSize = parseInt(document.getElementById('tile-size').value)
    || _options.defaultTileSize;
    const colorSequence = _grids[_currentGridIndex];
    const numTiles = colorSequence.length;
    const { rows, cols } = calculateGridDimensions(numTiles);

    // Set canvas dimensions
    canvas.width = cols * tileSize;
    canvas.height = rows * tileSize;

    // Clear the canvas
    context.clearRect(0, 0, canvas.width, canvas.height);

    // Draw the grid
    colorSequence.forEach((color, i) => {
      const col = i % cols;
      const row = Math.floor(i / cols);
      const x = col * tileSize;
      const y = row * tileSize;

      context.fillStyle = color;
      context.fillRect(x, y, tileSize, tileSize);

      // Optional: Add a border to each tile
      context.strokeStyle = '#ccc';
      context.strokeRect(x, y, tileSize, tileSize);
    });

    // Remove existing event listeners to prevent stacking
    canvas.onclick = null;

    // Add a single event listener to the canvas
    canvas.addEventListener('click', function(event) {
      const rect = canvas.getBoundingClientRect();
      const clickX = event.clientX - rect.left;
      const clickY = event.clientY - rect.top;

      const clickedCol = Math.floor(clickX / tileSize);
      const clickedRow = Math.floor(clickY / tileSize);
      const clickedIndex = clickedRow * cols + clickedCol;

      if (clickedIndex >= 0 && clickedIndex < colorSequence.length) {
        ...
      }
    });
  }
}
```

```
        const newColor = prompt("Enter new hex color (e.g., #00ff00):", colorSequence[clickedIndex]);
        if (newColor) {
            if (isValidHexColor(newColor)) {
                colorSequence[clickedIndex] = newColor;
                displayCurrentGrid(); // Redraw the grid
                showSuccess("Tile color updated");
            } else {
                showError('Please enter a valid hex color code (e.g., #00ff00).');
            }
        }
    });

    updateColorList();
    updateNavigationButtons();
} catch (error) {
    showError(`Error displaying grid: ${error.message}`);
    console.error("Error displaying grid:", error);
}
}

/**
 * Update the color list and display the compounded grid ID
 */
function updateColorList() {
    try {
        const colorList = document.getElementById('color-list');
        colorList.innerHTML = '';
        const colorSequence = _grids[_currentGridIndex];

        colorSequence.forEach((color, index) => {
            const listItem = document.createElement('li');
            listItem.textContent = `ID ${hexToID(color)}: ${color}`;
            colorList.appendChild(listItem);
        });

        const compoundedGridID = calculateCompoundedGridID(colorSequence);
        const gridIdElement = document.getElementById('current-grid-id');
        gridIdElement.textContent = `Compounded Grid ID: ${compoundedGridID}`;

        const gridPositionElement = document.getElementById('current-grid-position');
        gridPositionElement.textContent = `Grid ${_currentGridIndex + 1} of ${_grids.length}`;
    } catch (error) {
        showError(`Error updating color list: ${error.message}`);
        console.error("Error updating color list:", error);
    }
}

/**
 * Update the navigation buttons' disabled state

```

```
/*
function updateNavigationButtons() {
    try {
        const prevButton = document.getElementById('prev-grid');
        const nextButton = document.getElementById('next-grid');

        prevButton.disabled = _currentGridIndex === 0;
        nextButton.disabled = _currentGridIndex === _grids.length - 1;
    } catch (error) {
        console.error("Error updating navigation buttons:", error);
    }
}

/**
 * Split a compounded ID into tile IDs
 * @param {string} compoundedID - Compounded grid ID
 * @param {number} tileIDLength - Length of each tile ID
 * @returns {Array<string>} Array of tile IDs
 */
function splitCompoundedID(compoundedID, tileIDLength) {
    const tileIDs = [];
    let index = 0;
    while (index < compoundedID.length) {
        const tileID = compoundedID.substr(index, tileIDLength);
        tileIDs.push(tileID);
        index += tileIDLength;
    }
    return tileIDs;
}

/**
 * Regenerate grids from an array of compounded grid IDs
 * @param {Array<string>} compoundedIDs - Array of compounded grid IDs
 */
function regenerateGrids(compoundedIDs) {
    try {
        if (!compoundedIDs || !Array.isArray(compoundedIDs) ||
compoundedIDs.length === 0) {
            showError("No valid compounded IDs provided");
            return;
        }

        _grids = []; // Reset the grids array

        compoundedIDs.forEach((idString, index) => {
            const tileIDLength = 7; // Standard tile ID length
            let tileIDs = [];

            idString = idString.trim();

            if (/^\d+$/.test(idString)) {
                // It's a single large number; split it into tile IDs
                tileIDs = splitCompoundedID(idString, tileIDLength);
            } else {

```

```
        showError(`Invalid Compounded Grid ID format at index
${index}. Please enter a single large integer.`);
        throw new Error('Invalid Compounded Grid ID format');
    }

    const colorSequenceFromID = tileIDs.map(id => {
        try {
            const hexColor = idToHex(id);
            if (isValidHexColor(hexColor)) {
                return hexColor;
            } else {
                showError(`Invalid tile ID: ${id}`);
                return '#000000'; // Default to black
            }
        } catch (error) {
            console.error(`Error processing tile ID ${id}:`, error);
            return '#000000'; // Default to black
        }
    });
}

_grids.push(colorSequenceFromID);
});

_currentGridIndex = 0; // Start from the first grid
displayCurrentGrid();

showSuccess(`Successfully regenerated ${_grids.length} ${_grids.length
=== 1 ? 'grid' : 'grids'}`);
} catch (error) {
    showError(`Error regenerating grids: ${error.message}`);
    console.error("Error regenerating grids:", error);
}
}

/**
 * Add a new compounded ID input field
 */
function addNewCompoundedIDInput() {
    try {
        const inputsContainer = document.getElementById('compounded-id-
inputs');
        const newInputRow = document.createElement('div');
        newInputRow.className = 'compounded-id-input-row';

        const newInput = document.createElement('input');
        newInput.type = 'text';
        newInput.className = 'compounded-id-input';
        newInput.placeholder = 'Enter Compounded Grid ID';

        newInputRow.appendChild(newInput);
        inputsContainer.appendChild(newInputRow);
    } catch (error) {
        showError(`Error adding input field: ${error.message}`);
        console.error("Error adding input field:", error);
    }
}
```

```
        }

    }

    /**
     * Generate grids from input fields
     */
    function generateGridsFromInputs() {
        try {
            const inputFields = document.querySelectorAll('.compounded-id-input');
            const compoundedIDStrings = [];

            inputFields.forEach(input => {
                const idString = input.value.trim();
                if (idString !== '') {
                    compoundedIDStrings.push(idString);
                }
            });
        }

        if (compoundedIDStrings.length === 0) {
            showError('Please enter at least one Compounded Grid ID.');
            return;
        }

        regenerateGrids(compoundedIDStrings);
    } catch (error) {
        showError(`Error generating grids: ${error.message}`);
        console.error("Error generating grids:", error);
    }
}

/**
 * Process an uploaded file containing compounded grid IDs
 */
function processUploadedFile() {
    try {
        const fileInput = document.getElementById('file-input');
        const file = fileInput.files[0];

        if (!file) {
            showError("Please select a sequences.txt file before executing.");
            return;
        }

        const reader = new FileReader();

        reader.onload = function(event) {
            try {
                const content = event.target.result;
                const lines = content.split('\n').map(line => line.trim());

                const validIDs = lines.filter(line => /^d+$/.test(line)); // Only numeric lines
                if (validIDs.length > 0) {
                    regenerateGrids(validIDs); // Process the valid compound
                }
            }
        }
    }
}
```

IDs

```
        } else {
            showError("No valid compound IDs found in the file.");
        }
    } catch (error) {
    showError(`Error processing file content: ${error.message}`);
    console.error("Error processing file content:", error);
}
};

reader.onerror = function() {
    showError("Failed to read the file.");
};

reader.readAsText(file);
} catch (error) {
    showError(`Error processing file: ${error.message}`);
    console.error("Error processing file:", error);
}
}

/** 
 * Download the tile ID information
 */
function downloadTileID() {
try {
    if (_grids.length === 0 || _currentGridIndex >= _grids.length) {
        showError("No grid available to download");
        return;
    }

    const colorSequence = _grids[_currentGridIndex];
    const compoundedGridID = calculateCompoundedGridID(colorSequence);
    const fileContent = colorSequence.map((color, index) => `ID
${hexToID(color)}: ${color}`).join('\n');
    const fullContent = `Compounded Grid ID:
${compoundedGridID}\n\n${fileContent}`;

    // Create and download the file
    const blob = new Blob([fullContent], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);

    const a = document.createElement('a');
    a.href = url;
    a.download = 'color_sequence.txt';
    a.click();

    URL.revokeObjectURL(url);

    showSuccess("Tile ID information downloaded");
} catch (error) {
    showError(`Error downloading tile ID: ${error.message}`);
    console.error("Error downloading tile ID:", error);
}
}
```

```
}

/**
 * Download the current grid as an image
 */
function downloadGridImage() {
    try {
        const canvas = document.getElementById('grid-canvas');
        const image = canvas.toDataURL('image/png');

        // Create and download the image
        const a = document.createElement('a');
        a.href = image;
        a.download = `grid_${_currentGridIndex + 1}.png`;
        a.click();

        showSuccess("Grid image downloaded");
    } catch (error) {
        showError(`Error downloading grid image: ${error.message}`);
        console.error("Error downloading grid image:", error);
    }
}

/**
 * Show an error message
 * @param {string} message - Error message
 */
function showError(message) {
    // Remove any existing error messages
    const existingError = document.querySelector('.error-message');
    if (existingError) {
        existingError.remove();
    }

    // Remove any existing success messages
    const existingSuccess = document.querySelector('.success-message');
    if (existingSuccess) {
        existingSuccess.remove();
    }

    // Create and show the error message
    const container = document.getElementById('compounded-id-input-container');
    const errorElement = document.createElement('div');
    errorElement.className = 'error-message';
    errorElement.textContent = message;

    container.appendChild(errorElement);

    // Auto-remove after 5 seconds
    setTimeout(() => {
        if (document.body.contains(errorElement)) {
            errorElement.remove();
        }
    }, 5000);
}
```

```
        }, 5000);
    }

    /**
     * Show a success message
     * @param {string} message - Success message
     */
    function showSuccess(message) {
        // Remove any existing error messages
        const existingError = document.querySelector('.error-message');
        if (existingError) {
            existingError.remove();
        }

        // Remove any existing success messages
        const existingSuccess = document.querySelector('.success-message');
        if (existingSuccess) {
            existingSuccess.remove();
        }

        // Create and show the success message
        const container = document.getElementById('compounded-id-input-container');
        const successElement = document.createElement('div');
        successElement.className = 'success-message';
        successElement.textContent = message;

        container.appendChild(successElement);

        // Auto-remove after 5 seconds
        setTimeout(() => {
            if (document.body.contains(successElement)) {
                successElement.remove();
            }
        }, 5000);
    }

    /**
     * Get the current grids
     * @returns {Array<Array<string>>} The current grids
     */
    function getGrids() {
        return [..._grids];
    }

    /**
     * Get the current grid
     * @returns {Array<string>|null} The current grid or null if no grid is available
     */
    function getCurrentGrid() {
        if (_grids.length === 0 || _currentGridIndex >= _grids.length) {
            return null;
        }
    }
}
```

```
        return [..._grids[_currentGridIndex]];
    }

    /**
     * Set the current grid
     * @param {Array<string>} colorSequence - The color sequence for the grid
     * @param {boolean} [replace=false] - Whether to replace the current grid or
add a new one
     * @returns {boolean} Success status
    */
    function setCurrentGrid(colorSequence, replace = false) {
        if (!colorSequence || !Array.isArray(colorSequence)) {
            return false;
        }

        // Validate all colors
        for (const color of colorSequence) {
            if (!isValidHexColor(color)) {
                return false;
            }
        }

        if (replace && _grids.length > 0) {
            _grids[_currentGridIndex] = [...colorSequence];
        } else {
            _grids.push([...colorSequence]);
            _currentGridIndex = _grids.length - 1;
        }

        displayCurrentGrid();
        return true;
    }

    /**
     * Set the current grid index
     * @param {number} index - The index to set
     * @returns {boolean} Success status
    */
    function setCurrentGridIndex(index) {
        if (index < 0 || index >= _grids.length) {
            return false;
        }

        _currentGridIndex = index;
        displayCurrentGrid();
        return true;
    }

    /**
     * Convert a grid to a data URL
     * @param {number} [index] - Index of the grid to convert, defaults to current
grid
     * @returns {string|null} Data URL of the grid or null if no grid is available
    */
}
```

```
function gridToDataURL(index = null) {
    const gridIndex = index !== null ? index : _currentGridIndex;

    if (_grids.length === 0 || gridIndex < 0 || gridIndex >= _grids.length) {
        return null;
    }

    const colorSequence = _grids[gridIndex];
    const numTiles = colorSequence.length;
    const { rows, cols } = calculateGridDimensions(numTiles);
    const tileSize = parseInt(document.getElementById('tile-size').value) ||
        _options.defaultTileSize;

    // Create a temporary canvas
    const canvas = document.createElement('canvas');
    canvas.width = cols * tileSize;
    canvas.height = rows * tileSize;
    const context = canvas.getContext('2d');

    // Draw the grid
    colorSequence.forEach((color, i) => {
        const col = i % cols;
        const row = Math.floor(i / cols);
        const x = col * tileSize;
        const y = row * tileSize;

        context.fillStyle = color;
        context.fillRect(x, y, tileSize, tileSize);

        // Add a border to each tile
        context.strokeStyle = '#ccc';
        context.strokeRect(x, y, tileSize, tileSize);
    });

    return canvas.toDataURL('image/png');
}

// Public API
return {
    initialize,
    loadAlphabetSystem,
    showAlphabetSection,
    updateGrid,
    displayCurrentGrid,
    regenerateGrids,
    downloadTileID,
    downloadGridImage,
    getGrids,
    getCurrentGrid,
    setCurrentGrid,
    setCurrentGridIndex,
    gridToDataURL,
    hexToID,
    idToHex,
```

```

        calculateCompoundedGridID,
        isValidHexColor
    );
})();

// Automatically initialize when the DOM is ready
document.addEventListener('DOMContentLoaded', function() {
    // Look for the main container
    const mainElement = document.querySelector('main');
    if (mainElement) {
        OperatorFramework.Alphabet.initialize({
            containerId: mainElement.id || 'main'
        });
        console.log('Alphabet System initialized');
    } else {
        console.warn('Main element not found, Alphabet System not automatically initialized');
    }
});

```

## [92] integerDatabase/operator-build.js

- **Bytes:** 128872
- **Type:** text

```

/**
 * operator-build.js
 *
 * This module implements the Build System (Memory Slot Manager & Logic Framework)
 * for the Operator framework, providing memory management and logical operations.
 * It integrates with the operator-dropdown.js system.
 *
 * @module OperatorBuild
 * @version 1.0.0
 * @depends OperatorFramework.Dropdown
 */

// Access the shared namespace for Operator framework

/**
 * Build System for memory management and logical operations
 * @namespace OperatorBuild
 */
OperatorFramework.Build = (function() {
    'use strict';

    // Private module variables
    let _initialized = false;
    let _section = null;
    let _memoryManager = null;
    let _logicEngine = null;

```

```
let _iframe = null;
let _defaultOptions = {
    containerId: 'main', // ID of the container to add the build section to
    sectionId: 'build-system-section', // ID for the created section element
    iframeId: 'build-system-iframe', // ID for the iframe element
    iframeHeight: '800px' // Default height for the iframe
};
let _options = {};
let _loadCallbacks = [];

/** 
 * Initialize the Build System
 * @param {Object} options - Configuration options
 * @param {string} [options.containerId='main'] - ID of the container to add the build section to
 * @param {string} [options.sectionId='build-system-section'] - ID for the created section element
 * @param {string} [options.iframeId='build-system-iframe'] - ID for the iframe element
 * @param {string} [options.iframeHeight='800px'] - Height for the iframe
 * @returns {boolean} Success status
*/
function initialize(options = {}) {
    if (_initialized) {
        console.warn('OperatorBuild is already initialized');
        return false;
    }

    // Merge options with defaults
    _options = { ..._defaultOptions, ...options };

    console.log('OperatorBuild initialized with default options');
    _initialized = true;

    // Register with dropdown system if available
    if (OperatorFramework.Dropdown) {
        OperatorFramework.Dropdown.addSystem({
            id: 'build',
            name: 'Memory Slot Manager & Logic Framework',
            description: 'Manage memory slots and perform logical operations',
            loadFunction: loadBuildSystem
        });
        console.log('Build System registered with OperatorFramework.Dropdown');
    } else {
        console.warn('OperatorFramework.Dropdown not available, Build System not registered');
    }

    return true;
}

/** 
 * Load the Build System UI and functionality
*/
```

```
* @returns {boolean} Success status
*/
function loadBuildSystem() {
    if (!initialized) {
        console.error('OperatorBuild not initialized, call initialize() first');
        return false;
    }

    // Check if already loaded
    if (_section && document.getElementById(_options.sectionId)) {
        showBuildSection();
        return true;
    }

    // Get container
    //const container = document.getElementById(_options.containerId);
    // Use:
    const container = document.querySelector('main');
    if (!container) {
        console.error(`Container with ID "${_options.containerId}" not found`);
        return false;
    }

    // Create Build System section
    _section = document.createElement('section');
    _section.id = _options.sectionId;
    _section.className = 'operation-section';

    // Create header
    const header = document.createElement('h2');
    header.textContent = 'Memory Slot Manager & Logic Framework';
    _section.appendChild(header);

    // Create container div
    const operationContainer = document.createElement('div');
    operationContainer.className = 'operation-container';
    _section.appendChild(operationContainer);

    // Create loading indicator
    const loadingIndicator = document.createElement('div');
    loadingIndicator.id = 'build-loading-indicator';
    loadingIndicator.style.textAlign = 'center';
    loadingIndicator.style.padding = '20px';
    loadingIndicator.innerHTML =
        `

Loading Memory Slot Manager & Logic Framework...</div>
        <div class="spinner" style="
            border: 4px solid rgba(0, 0, 0, 0.1);
            border-left-color: #4B5320;
            border-radius: 50%;
            width: 30px;
            height: 30px;
        `


```

```
        animation: spin 1s linear infinite;
        margin: 0 auto;
    "></div>
<style>
    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
</style>
`;
operationContainer.appendChild(loadingIndicator);

// Create iframe (initially hidden)
_iframe = document.createElement('iframe');
_iframe.id = _optionsiframeId;
_iframe.style.width = '100%';
_iframe.style.height = _optionsiframeHeight;
_iframe.style.border = 'none';
_iframe.style.display = 'none';
operationContainer.appendChild(_iframe);

// Add to container
container.appendChild(_section);

// Show the section
showBuildSection();

// Create the iframe content
createBuildSystemIframe();

console.log('Build System loaded successfully');
return true;
}

/**
 * Show the Build System section and hide other sections
 */
function showBuildSection() {
    if (!_section) return;

    // Hide all sections
    const allSections = document.querySelectorAll('.operation-section');
    allSections.forEach(section => {
        section.classList.remove('active');
    });

    // Show Build section
    _section.classList.add('active');
}

/**
 * Create and populate the Build System iframe
 */
function createBuildSystemIframe() {
```

```
if (!_iframe) return;

// Generate iframe content
const iframeContent = generateIframeHTML();

// Set up iframe load event
_iframe.onload = function() {
    // Hide loading indicator
    const loadingIndicator = document.getElementById('build-loading-indicator');
    if (loadingIndicator) {
        loadingIndicator.style.display = 'none';
    }

    // Show iframe
    _iframe.style.display = 'block';

    // Access iframe contents
    const iframeWindow = _iframe.contentWindow;
    const iframeDocument = _iframe.contentDocument ||
iframeWindow.document;

    // Expose parent functions to iframe
    iframeWindow.parentAPI = createParentAPI();

    // Add event listeners for iframe-to-parent communication
    setupIframeCommunication(iframeWindow);

    // Store references to memory manager and logic engine
    if (iframeWindow.memoryManager) {
        _memoryManager = iframeWindow.memoryManager;
    }

    if (iframeWindow.logicEngine) {
        _logicEngine = iframeWindow.logicEngine;
    }

    // Execute any queued load callbacks
    for (const callback of _loadCallbacks) {
        try {
            callback(_memoryManager, _logicEngine, iframeWindow);
        } catch (error) {
            console.error('Error executing load callback:', error);
        }
    }

    // Clear load callbacks
    _loadCallbacks = [];

    console.log('Build System iframe loaded and initialized');
};

// Load content into iframe
_iframe.srcdoc = iframeContent;
```

```
}

/** 
 * Create API object for parent-to-iframe communication
 * @returns {Object} API object
 */
function createParentAPI() {
    return {
        executeCommand: function(command) {
            return executeCommand(command);
        },
        notifyParent: function(message, type) {
            console.log(`Message from iframe: ${message}`, type);
        },
        getParentData: function() {
            return {
                timestamp: new Date().toISOString(),
                origin: window.location.origin,
                framework: 'OperatorFramework',
                version: '1.0.0'
            };
        }
    };
}

/** 
 * Set up communication between parent and iframe
 * @param {Window} iframeWindow - The iframe window object
 */
function setupIframeCommunication(iframeWindow) {
    // Listen for messages from iframe
    window.addEventListener('message', function(event) {
        // Verify origin
        if (event.source !== iframeWindow) return;

        const { action, data } = event.data || {};

        if (!action) return;

        switch (action) {
            case 'execute-command':
                if (data && data.command) {
                    const result = executeCommand(data.command);
                    // Send result back to iframe
                    iframeWindow.postMessage({
                        action: 'command-result',
                        data: { result }
                    }, '*');
                }
                break;

            case 'notify-parent':
                if (data && data.message) {
                    console.log(`Notification from iframe: ${data.message}`);
                }
        }
    });
}
```

```
data.type);
    }
    break;

    case 'request-data':
        // Send data to iframe
        iframeWindow.postMessage({
            action: 'parent-data',
            data: {
                timestamp: new Date().toISOString(),
                origin: window.location.origin,
                framework: 'OperatorFramework',
                version: '1.0.0'
            }
        }, '*');
        break;
    }
});

}

/**
 * Execute a command in the iframe
 * @param {string} command - Command to execute
 * @returns {Promise<any>} Command result
 */
function executeCommand(command) {
    return new Promise((resolve, reject) => {
        if (!_iframe || !_iframe.contentWindow) {
            reject(new Error('Iframe not available'));
            return;
        }

        try {
            // Process the command
            const iframeWindow = _iframe.contentWindow;

            if (iframeWindow.submitCommand) {
                // For terminal commands
                iframeWindow.submitCommand(command);
                resolve(true);
            } else if (iframeWindow.eval) {
                // Direct evaluation (less secure but more flexible)
                const result = iframeWindow.eval(command);
                resolve(result);
            } else {
                reject(new Error('No command execution method available in
iframe')));
            }
        } catch (error) {
            reject(error);
        }
    });
}
```

```
/**  
 * Register a callback to be executed when the iframe is loaded  
 * @param {Function} callback - Callback function(memoryManager, logicEngine,  
iframeWindow)  
 */  
function onLoad(callback) {  
    if (typeof callback !== 'function') {  
        console.error('onLoad requires a function parameter');  
        return;  
    }  
  
    if (_memoryManager && _logicEngine && _iframe && _iframe.contentWindow) {  
        // If already loaded, execute immediately  
        try {  
            callback(_memoryManager, _logicEngine, _iframe.contentWindow);  
        } catch (error) {  
            console.error('Error executing load callback:', error);  
        }  
    } else {  
        // Otherwise, queue for later execution  
        _loadCallbacks.push(callback);  
    }  
}  
  
/**  
 * Generate the HTML content for the iframe  
 * @returns {string} HTML content  
 */  
function generateIframeHTML() {  
    return `<!DOCTYPE html>  
    <html lang="en">  
        <head>  
            <meta charset="UTF-8">  
            <meta name="viewport" content="width=device-width, initial-scale=1.0">  
            <title>Memory Slot Manager & Logic Framework</title>  
            <style>  
                :root {  
                    --primary-color: #0078d7;  
                    --secondary-color: #4a4a4a;  
                    --background-color: #f9f9f9;  
                    --terminal-bg: #1e1e1e;  
                    --terminal-text: #f0f0f0;  
                    --border-color: #dbdbdb;  
                    --highlight-color: #e6f2ff;  
                }  
  
                * {  
                    box-sizing: border-box;  
                    margin: 0;  
                    padding: 0;  
                    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
                }  
  
                body {  
                    /* Styling for the body */  
                }  
            </style>  
        </head>  
        <body>  
            <div id="app">  
                <h1>Memory Slot Manager & Logic Framework</h1>  
                <p>This is a demonstration of the Memory Slot Manager & Logic Framework.  
                It uses an iframe to host a separate application, which can be loaded  
                via the onLoad callback.  
            </div>  
        </body>  
    </html>
```

```
background-color: var(--background-color);
line-height: 1.6;
color: var(--secondary-color);
padding: 20px;
max-width: 1200px;
margin: 0 auto;
}

header {
    text-align: center;
    margin-bottom: 30px;
    padding: 20px;
    background-color: white;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

h1 {
    color: var(--primary-color);
    margin-bottom: 10px;
}

.system-info {
    font-size: 0.9em;
    color: var(--secondary-color);
}

.main-container {
    display: flex;
    flex-direction: column;
    gap: 20px;
}

@media (min-width: 992px) {
    .main-container {
        flex-direction: row;
    }

    .terminal-section {
        flex: 1;
    }

    .results-section {
        flex: 1;
    }
}

.section {
    background-color: white;
    border-radius: 5px;
    padding: 20px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}
```

```
.section-title {  
    font-size: 1.2em;  
    margin-bottom: 15px;  
    padding-bottom: 10px;  
    border-bottom: 1px solid var(--border-color);  
    color: var(--primary-color);  
    display: flex;  
    justify-content: space-between;  
    align-items: center;  
}  
  
.section-title-actions {  
    display: flex;  
    gap: 10px;  
}  
  
.action-button {  
    padding: 3px 8px;  
    font-size: 0.8em;  
    background-color: var(--primary-color);  
    color: white;  
    border: none;  
    border-radius: 3px;  
    cursor: pointer;  
}  
  
.action-button:hover {  
    background-color: #005a9e;  
}  
  
.terminal {  
    background-color: var(--terminal-bg);  
    color: var(--terminal-text);  
    padding: 15px;  
    border-radius: 5px;  
    font-family: 'Consolas', 'Courier New', monospace;  
    height: 350px;  
    overflow-y: auto;  
    margin-bottom: 15px;  
}  
  
.terminal-input {  
    display: flex;  
    margin-bottom: 10px;  
}  
  
.terminal-input span {  
    margin-right: 10px;  
    color: #4caf50;  
}  
  
.commandInput {  
    width: 100%;  
    padding: 10px;  
}
```

```
font-family: 'Consolas', 'Courier New', monospace;
margin-bottom: 10px;
border: 1px solid var(--border-color);
border-radius: 5px;
}

.button {
    background-color: var(--primary-color);
    color: white;
    border: none;
    padding: 10px 15px;
    border-radius: 5px;
    cursor: pointer;
    font-size: 0.9em;
    transition: background-color 0.3s;
}

.button:hover {
    background-color: #005a9e;
}

.button-secondary {
    background-color: #6c757d;
}

.button-secondary:hover {
    background-color: #5a6268;
}

#multilineInput {
    width: 100%;
    min-height: 100px;
    padding: 10px;
    font-family: 'Consolas', 'Courier New', monospace;
    margin-bottom: 10px;
    border: 1px solid var(--border-color);
    border-radius: 5px;
    display: none;
    white-space: pre;
    overflow-wrap: normal;
    overflow-x: auto;
}

.memory-display {
    background-color: white;
    border: 1px solid var(--border-color);
    border-radius: 5px;
    padding: 15px;
    height: 500px;
    overflow-y: auto;
}

.memory-file {
    margin-bottom: 20px;
```

```
}

.memory-file-title {
    font-weight: bold;
    margin-bottom: 5px;
    padding-bottom: 5px;
    border-bottom: 1px solid var(--border-color);
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.memory-file-actions {
    display: flex;
    gap: 5px;
}

.file-action {
    font-size: 0.7em;
    padding: 2px 5px;
    background-color: var(--primary-color);
    color: white;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}

.file-action:hover {
    background-color: #005a9e;
}

.memory-slot {
    padding: 8px;
    border-bottom: 1px solid #f0f0f0;
}

.memory-slot:hover {
    background-color: var(--highlight-color);
}

.memory-slot-title {
    font-weight: bold;
    margin-bottom: 5px;
}

.memory-slot-value {
    white-space: pre-wrap;
    font-family: 'Consolas', 'Courier New', monospace;
    padding: 5px;
    background-color: #f8f8f8;
    border-radius: 3px;
    font-size: 0.9em;
}
```

```
.quick-commands {  
    display: flex;  
    flex-wrap: wrap;  
    gap: 10px;  
    margin-top: 15px;  
}  
  
.quick-command {  
    background-color: #f0f0f0;  
    padding: 5px 10px;  
    border-radius: 3px;  
    font-size: 0.8em;  
    cursor: pointer;  
    transition: background-color 0.3s;  
}  
  
.quick-command:hover {  
    background-color: #e0e0e0;  
}  
  
.modal {  
    display: none;  
    position: fixed;  
    z-index: 1;  
    left: 0;  
    top: 0;  
    width: 100%;  
    height: 100%;  
    overflow: auto;  
    background-color: rgba(0,0,0,0.5);  
}  
  
.modal-content {  
    background-color: #fefefe;  
    margin: 15% auto;  
    padding: 20px;  
    border: 1px solid #888;  
    width: 80%;  
    max-width: 600px;  
    border-radius: 5px;  
}  
  
.close-modal {  
    color: #aaa;  
    float: right;  
    font-size: 28px;  
    font-weight: bold;  
    cursor: pointer;  
}  
  
.close-modal:hover {  
    color: black;  
}
```

```
.progress-bar-container {
    width: 100%;
    background-color: #e0e0e0;
    border-radius: 3px;
    margin: 10px 0;
}

.progress-bar {
    height: 20px;
    background-color: var(--primary-color);
    border-radius: 3px;
    width: 0%;
    transition: width 0.3s;
}

#generationStatus {
    margin-top: 10px;
    font-size: 0.9em;
}

/* File system styles */
.files-section {
    margin-top: 20px;
}

.file-system-actions {
    display: flex;
    gap: 10px;
    margin-bottom: 10px;
}

/* Radio button styling */
.radio-group {
    margin: 15px 0;
}

.radio-option {
    margin-bottom: 10px;
    display: flex;
    align-items: flex-start;
}

.radio-option input[type="radio"] {
    margin-top: 3px;
    margin-right: 10px;
}

.radio-option .option-label {
    font-weight: bold;
}

.radio-option .option-description {
    font-size: 0.85em;
    color: #666;
```

```
        margin-top: 2px;
    }

/* File input styling */
.file-input-container {
    margin: 15px 0;
}

.file-input-label {
    font-weight: bold;
    margin-bottom: 5px;
    display: block;
}

.file-input {
    width: 100%;
    padding: 8px;
    border: 1px solid var(--border-color);
    border-radius: 4px;
}

/* Logic engine styles */
.nav-tabs {
    display: flex;
    border-bottom: 1px solid var(--border-color);
    margin-bottom: 15px;
}

.nav-tab {
    padding: 8px 15px;
    cursor: pointer;
    border: 1px solid transparent;
    border-bottom: none;
    border-radius: 5px 5px 0 0;
    margin-right: 5px;
    background-color: #f8f8f8;
}

.nav-tab.active {
    background-color: white;
    border-color: var(--border-color);
    border-bottom: 1px solid white;
    margin-bottom: -1px;
    font-weight: bold;
}

.tab-content {
    display: none;
}

.tab-content.active {
    display: block;
}
```

```
#logicTerminal {
    background-color: var(--terminal-bg);
    color: var(--terminal-text);
    padding: 15px;
    border-radius: 5px;
    font-family: 'Consolas', 'Courier New', monospace;
    height: 300px;
    overflow-y: auto;
    margin-bottom: 15px;
}

@media (max-width: 768px) {
    body {
        padding: 10px;
    }

    .terminal {
        height: 250px;
    }

    .memory-display {
        height: 300px;
    }

    .modal-content {
        width: 95%;
        margin: 10% auto;
    }

    .file-system-actions {
        flex-direction: column;
    }
}

</style>
</head>
<body>
    <header>
        <h1>INTEGRATED MEMORY & LOGIC FRAMEWORK</h1>
        <div class="system-info">
            <div id="systemTime"></div>
            <div>MEMORY MANAGER & LOGIC ENGINE LOADED</div>
            <div>READY FOR INPUT</div>
        </div>
    </header>

    <div class="main-container">
        <div class="terminal-section">
            <div class="section">
                <div class="nav-tabs">
                    <div class="nav-tab active" data-tab="memory">Memory
Manager</div>
                    <div class="nav-tab" data-tab="logic">Logic
Engine</div>
                </div>
            </div>
        </div>
    </div>
```

```
<div id="memoryTab" class="tab-content active">
    <div class="section-title">
        Command Terminal
        <div class="section-title-actions">
            <button class="action-button"
onlick="clearTerminal()">Clear</button>
            </div>
        </div>
        <div id="terminal" class="terminal"></div>
        <input type="text" id="commandInput"
placeholder="Enter command (type 'help' for list of commands)">
        <textarea id="multilineInput" placeholder="Enter
multi-line content (e.g. for assign or search)" wrap="off"></textarea>
        <div class="quick-commands">
            <span class="quick-command"
onlick="insertCommand('help')">help</span>
            <span class="quick-command"
onlick="insertCommand('display')">display</span>
            <span class="quick-command"
onlick="insertCommand('assign')">assign</span>
            <span class="quick-command"
onlick="insertCommand('read')">read</span>
            <span class="quick-command"
onlick="insertCommand('search')">search</span>
            <span class="quick-command"
onlick="insertCommand('generate')">generate</span>
            <span class="quick-command"
onlick="insertCommand('save_all')">save_all</span>
            <span class="quick-command"
onlick="insertCommand('load_all')">load_all</span>
        </div>
        <div style="display: flex; gap: 10px; margin-top:
10px;">
            <button id="submitCommand" class="button">Execute
Command</button>
            <button id="submitMultiline" class="button"
style="display: none;">Submit</button>
            <button id="cancelMultiline" class="button button-
secondary" style="display: none;">Cancel</button>
        </div>
    </div>

    <div id="logicTab" class="tab-content">
        <div class="section-title">
            Proof by Contradiction Engine
            <div class="section-title-actions">
                <button class="action-button"
onlick="clearLogicTerminal()">Clear</button>
                </div>
            </div>
            <div id="logicTerminal" class="terminal"></div>
            <input type="text" id="logicCommandInput"
placeholder="Enter instruction (type 'help' for available instructions)">
        </div>
    </div>
```

```
        <div class="quick-commands">
            <span class="quick-command"
        onclick="insertLogicCommand('help')">help</span>
            <span class="quick-command"
        onclick="insertLogicCommand('ASSERT')">ASSERT</span>
            <span class="quick-command"
        onclick="insertLogicCommand('NOT')">NOT</span>
            <span class="quick-command"
        onclick="insertLogicCommand('AND')">AND</span>
            <span class="quick-command"
        onclick="insertLogicCommand('OR')">OR</span>
            <span class="quick-command"
        onclick="insertLogicCommand('IMPLIES')">IMPLIES</span>
            <span class="quick-command"
        onclick="insertLogicCommand('CONTRADICTION')">CONTRADICTION</span>
            </div>
        <div style="display: flex; gap: 10px; margin-top: 10px;">
            <button id="submitLogicCommand"
        class="button">Execute Instruction</button>
            <button id="checkContradictions"
        class="button">Check Contradictions</button>
            <button id="clearStatements" class="button button-secondary">Clear Statements</button>
            <button id="generateStatements"
        class="button">Generate Statements</button>
            </div>
        </div>
    </div>

        <div class="section files-section">
            <div class="section-title">File System</div>
            <div class="file-system-actions">
                <button class="button"
        onclick="saveAllMemorySlots()">Save All Files</button>
                <button class="button"
        onclick="loadMemorySlotsFromFile()">Load Files</button>
                <button class="button button-secondary"
        onclick="clearAllMemorySlots()">Clear All Files</button>
            </div>
        </div>
    </div>

        <div class="results-section">
            <div class="section">
                <div class="section-title">
                    Memory Slots
                <div class="section-title-actions">
                    <button class="action-button"
        onclick="updateMemoryDisplay()">Refresh</button>
                </div>
            </div>
            <div id="memoryDisplay" class="memory-display">
                <div style="text-align: center; color: #888; margin-
```

```
top: 20px;">
        No memory slots available.
        <br><br>
        Use the terminal to add memory slots.
    </div>
</div>
</div>
</div>
</div>
</div>

<div id="generationModal" class="modal">
    <div class="modal-content">
        <span class="close-modal"
onlick="document.getElementById('generationModal').style.display='none'">&times;
        </span>
        <h2>Generating Combinations</h2>

        <div class="radio-group">
            <div class="radio-option">
                <input type="radio" id="generateSingleFile"
name="generateType" value="singleFile" checked>
                <div>
                    <div class="option-label">Single File with
Multiple Slots</div>
                    <div class="option-description">Store all
combinations in one file with each combination as a separate slot</div>
                </div>
            </div>
            <div class="radio-option">
                <input type="radio" id="generateMultiFile"
name="generateType" value="multiFile">
                <div>
                    <div class="option-label">One File Per
Combination</div>
                    <div class="option-description">Create a separate
file for each generated combination</div>
                </div>
            </div>
        </div>
    </div>

    <div id="fileInputContainer" class="file-input-container">
        <div class="file-input-label">Enter Filename for Single
File Mode:</div>
        <input type="text" id="fileNameInput" class="file-input"
value="combinations" placeholder="Enter filename (e.g. 3, combinations, etc.)">
    </div>

    <div class="progress-bar-container">
        <div id="generationProgress" class="progress-bar"></div>
    </div>
    <div id="generationStatus">Ready to generate</div>
    <div style="margin-top: 20px; display: flex; gap: 10px; flex-
wrap: wrap;">
        <button id="startGenerationBtn" class="button">Start

```

```
Generation</button>
                <button id="downloadGeneratedBtn" class="button" disabled>Download Results</button>
                <button id="importGeneratedBtn" class="button" disabled>Import to Memory Slots</button>
                <button class="button button-secondary" onclick="document.getElementById('generationModal').style.display='none'">Close</button>
            </div>
        </div>
    </div>

    <div id="statementsGenerationModal" class="modal">
        <div class="modal-content">
            <span class="close-modal" onclick="document.getElementById('statementsGenerationModal').style.display='none'>&times;</span>
            <h2>Generate Statements</h2>

            <div class="file-input-container">
                <div class="file-input-label">Number of statements to generate:</div>
                <input type="number" id="numStatementsInput" class="file-input" value="5" min="1" max="100">
            </div>

            <div style="margin-top: 20px; display: flex; gap: 10px; flex-wrap: wrap;">
                <button id="startStatementsGenBtn" class="button">Generate Statements</button>
                <button class="button button-secondary" onclick="document.getElementById('statementsGenerationModal').style.display='none'">Cancel</button>
            </div>
        </div>
    </div>

    <div id="helpModal" class="modal">
        <div class="modal-content">
            <span class="close-modal" onclick="document.getElementById('helpModal').style.display='none'">&times;</span>
            <h2>Available Commands</h2>
            <pre style="white-space: pre-wrap; max-height: 400px; overflow-y: auto; margin-top: 10px;">
AVAILBLE COMMANDS:
-----
assign &lt;fileNumber&gt; &lt;slotNumber&gt;
    &lt;value&gt;          : Assign a multi-line value to a slot
                           (Enter your multi-line value in the text area)

read &lt;fileNumber&gt; &lt;slotNumber&gt;
    : Read the value from a slot

last_slot &lt;fileNumber&gt;

```

```
: Get the last slot number in a file

search
<value> : Search for a value across all slots

call <fileNumber> <startSlot> <endSlot>
      : Call values from a range of slots

export <fileNumber> : Export a file's memory slots to a text file

generate <n> : Generate all combinations with n cells
                using the standard character set
                Results are ready to use with this application

                Generation Options:
                - Single File: All combinations in one file with multiple
slots
                - One File Per Combination: Each combination in its own
file

custom_generate <n> : Generate combinations with custom character set

display : Show all memory slots
clear : Clear the terminal
help : Display this help message
```

```
save_json <fileNumber> : Save a specific file to JSON
save_all : Save all memory slots to a ZIP archive
load_json : Load memory slots from a JSON file
load_all : Load memory slots from a ZIP archive
```

#### File System Operations:

---

The file system supports downloading all your data as a ZIP archive and uploading it again for continuous use. This provides an automatic way to save your work and continue where you left off.

Generated files are automatically structured to be used with the memory slot manager system without any additional processing.

```
</pre>
</div>
</div>

<div id="logicHelpModal" class="modal">
  <div class="modal-content">
    <span class="close-modal"
  onclick="document.getElementById('logicHelpModal').style.display='none'">&times;
</span>
    <h2>Proof by Contradiction Engine Help</h2>
    <pre style="white-space: pre-wrap; max-height: 400px;
overflow-y: auto; margin-top: 10px;">
Proof by Contradiction Engine Help
-----
1. Add Instruction:
```

- ASSERT &lt;statement&gt;
  - NOT &lt;statement&gt;
  - AND &lt;statement1&gt; &lt;statement2&gt;
  - OR &lt;statement1&gt; &lt;statement2&gt;
  - IMPLIES &lt;statement1&gt; &lt;statement2&gt;
  - CONTRADICTION &lt;statement1&gt; &lt;statement2&gt;
2. Clear Statements: Clears all statements from the current framework.
  3. Check Contradictions: Checks for contradictions within the current framework using defined rules.
  4. Generate Statements: Automatically generates a framework of statements.
  5. Help: Displays this help message.

#### Example:

```
ASSERT Statement1
NOT Statement1
CONTRADICTION Statement1 NOT Statement1
(This will detect a contradiction based on the statement_not_statement rule)
</pre>
</div>
</div>

<script>
    class MemorySlotManager {
        constructor() {
            this.memorySlots = {};
        }

        assignSlot(fileNumber, slotNumber, value) {
            if (!fileNumber || !slotNumber || !value) {
                return "Error: Invalid input parameters.";
            }

            if (this.valueExists(value)) {
                return "Error: The value already exists in another
memory slot.";
            }

            if (!this.memorySlots[fileNumber]) {
                this.memorySlots[fileNumber] = {};
            }

            this.memorySlots[fileNumber][slotNumber] = value;
            return `Assigned value to slot ${slotNumber} in file
${fileNumber}.`;
        }

        readSlot(fileNumber, slotNumber) {
            if (this.memorySlots[fileNumber] &&
this.memorySlots[fileNumber][slotNumber] !== undefined) {
                return this.memorySlots[fileNumber][slotNumber];
            }
            return `Slot ${slotNumber} not found in file
${fileNumber}.`;
        }
    }
}
```

```
getLastSlotNumber(fileNumber) {
    if (this.memorySlots[fileNumber] &&
Object.keys(this.memorySlots[fileNumber]).length > 0) {
        try {
            const slots = [];
            for (const slot in this.memorySlots[fileNumber]) {
                if (/^\d+$/.test(slot)) {
                    slots.push(parseInt(slot));
                }
            }

            if (slots.length > 0) {
                return Math.max(...slots);
            }
        } catch (e) {
            // Ignore exceptions
        }
    }
    return null;
}

searchValue(value) {
    const results = [];
    const searchValueLower = value.toLowerCase();

    for (const file in this.memorySlots) {
        for (const slot in this.memorySlots[file]) {
            const val = this.memorySlots[file][slot];
            if (val.toLowerCase().includes(searchValueLower))
{
                results.push({
                    file,
                    slot,
                    value: val
                });
            }
        }
    }

    return results;
}

valueExists(value) {
    const valueLower = value.toLowerCase();

    for (const file in this.memorySlots) {
        for (const slot in this.memorySlots[file]) {
            if (this.memorySlots[file][slot].toLowerCase() ===
valueLower) {
                return true;
            }
        }
    }
}
```

```
        return false;
    }

    callSlotRange(fileNumber, startSlot, endSlot) {
        if (!this.memorySlots[fileNumber]) {
            return `File \${fileNumber} not found.`;
        }

        const results = [];
        for (let slot = startSlot; slot <= endSlot; slot++) {
            const slotStr = slot.toString();
            if (this.memorySlots[fileNumber][slotStr] !==
undefined) {
                results.push(`Slot
\${slotStr}:\\n\${this.memorySlots[fileNumber][slotStr]}`);
            } else {
                results.push(`Slot \${slotStr} not found.`);
            }
        }

        return results.join('\\n\\n');
    }

    exportToTextFile(fileNumber) {
        if (!this.memorySlots[fileNumber]) {
            return `Error: File \${fileNumber} not found.`;
        }

        try {
            let content = '';
            const slots = this.memorySlots[fileNumber];
            let slotKeys = [];

            try {
                const numericKeys = [];
                for (const key in slots) {
                    if (/^\d+$/.test(key)) {
                        numericKeys.push(parseInt(key));
                    }
                }
                numericKeys.sort((a, b) => a - b);
                slotKeys = numericKeys.map(key => key.toString());
            } catch (e) {
                // If conversion fails, sort alphabetically
                slotKeys = Object.keys(slots).sort();
            }

            for (const slot of slotKeys) {
                const value = slots[slot];
                // Format: SLOT <number>
                content += `SLOT \${slot}\\n\${value}\\n-----
-----\\n`;
            }
        }
    }
}
```

```
// Create a download link
const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `file\${fileNumber}.txt`;
    a.click();
    URL.revokeObjectURL(url);

    return `Exported file \${fileNumber} to
file\${fileNumber}.txt`;
} catch (e) {
    return `Error exporting to text file:
\${e.message}`;
}

saveToJSON(fileNumber) {
    try {
        if (fileNumber && !this.memorySlots[fileNumber]) {
            return `Error: File \${fileNumber} not found.`;
        }

        let jsonData;
        let filename;

        if (fileNumber) {
            jsonData = this.memorySlots[fileNumber];
            filename = `file\${fileNumber}.json`;
        } else {
            jsonData = this.memorySlots;
            filename = 'memory_slots.json';
        }

        const jsonString = JSON.stringify(jsonData, null, 2);
        const blob = new Blob([jsonString], { type:
'application/json' });

        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = filename;
        a.click();
        URL.revokeObjectURL(url);

        return `Memory slots saved to \${filename}\`;
    } catch (e) {
        return `Error saving to JSON: \${e.message}\`;
    }
}

loadJSON(jsonData) {
    try {
```

```
const content = JSON.parse(jsonData);

if (typeof content === 'object' && content !== null) {
    // Check if this is our expected format
    let isFullStructure = false;
    for (const file in content) {
        if (typeof content[file] === 'object' &&
content[file] !== null) {
            isFullStructure = true;
            break;
        }
    }

    if (isFullStructure) {
        // Merge with existing memory slots
        for (const file in content) {
            if (!this.memorySlots[file]) {
                this.memorySlots[file] = {};
            }

            for (const slot in content[file]) {
                this.memorySlots[file][slot] =
content[file][slot];
            }
        }
    } else {
        // Assume it's a single file's slots
        const fileNumber = '1'; // Default file number
        if (!this.memorySlots[fileNumber]) {
            this.memorySlots[fileNumber] = {};
        }

        for (const slot in content) {
            this.memorySlots[fileNumber][slot] =
content[slot];
        }
    }
}

return 'Successfully loaded JSON data.';
}

return 'Error: Invalid JSON structure.';
} catch (e) {
    return `Error loading JSON: \${e.message}`;
}
}

// Save all memory slots as a ZIP file
saveAllAsZip() {
    try {
        // Create a direct download of all slots as JSON for
simpler implementation
        const jsonData = JSON.stringify(this.memorySlots,
null, 2);
    }
}
```

```
const blob = new Blob([jsonData], { type:  
'application/json' });  
  
const url = URL.createObjectURL(blob);  
const a = document.createElement('a');  
a.href = url;  
a.download = 'all_memory_slots.json';  
a.click();  
URL.revokeObjectURL(url);  
  
return 'All memory slots saved to  
all_memory_slots.json';  
} catch (e) {  
    return `\`Error saving memory slots: \${e.message}\``;  
}  
}  
  
// Load memory slots from a JSON file (simplified without ZIP)  
loadFromFile(file) {  
    return new Promise((resolve, reject) => {  
        const reader = new FileReader();  
  
        reader.onload = (e) => {  
            try {  
                const result = this.loadJSON(e.target.result);  
                resolve(result);  
            } catch (error) {  
                reject(`\`Error parsing file:  
\${error.message}\``);  
            }  
        };  
  
        reader.onerror = () => {  
            reject('Error reading file');  
        };  
  
        reader.readAsText(file);  
    });  
}  
  
generateCharSet() {  
    let chars =  
        "abcdefghijklmnopqrstuvwxyz" +  
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ" +  
        "0123456789" +  
        "!@#$%^&*()-_=+[{}]|;:,.<>/?" +  
        "\\\t\\\\n\\\\r" +  
        "~\\\\\"\\\\\\\\";  
  
    // Ensure exactly 100 characters  
    if (chars.length > 100) {  
        chars = chars.substring(0, 100);  
    } else if (chars.length < 100) {  
        // Pad with additional characters if needed  
        while (chars.length < 100) {  
            chars += " ";  
        }  
    }  
}
```

```
        chars += '?';
    }
}

return chars;
}

async generateCombinations(chars, n, generateType, fileName,
progressCallback) {
    // The number of possible combinations is (k+1)^n
    const k = chars.length - 1;
    const nbrComb = Math.pow(k + 1, n);

    let fileContent = '';

    if (nbrComb > 1000000) {
        progressCallback(0, "Warning: Generating a large
number of combinations. This may take a while.");
    }

    // Clear any existing memory slots that would be affected
    if (generateType === 'singleFile') {
        // Clear just the target file
        this.memorySlots[fileName] = {};
    }

    // To avoid blocking the UI for too long, break up the
    work
    const batchSize = 10000; // Process in batches to keep UI
    responsive
    const totalBatches = Math.ceil(nbrComb / batchSize);

    for (let batch = 0; batch < totalBatches; batch++) {
        await new Promise(resolve => setTimeout(resolve, 0));
    // Allow UI to update

        const startRow = batch * batchSize;
        const endRow = Math.min((batch + 1) * batchSize,
nbrComb);

        for (let row = startRow; row < endRow; row++) {
            const id = row + 1;
            let combination = '';

            for (let col = n - 1; col >= 0; col--) {
                const rdiv = Math.pow(k + 1, col);
                const cell = Math.floor(row / rdiv) % (k + 1);
                combination += chars[cell];
            }

            // Handle different generation types
            if (generateType === 'singleFile') {
                // Store all combinations in a single file
                with the given name
            }
        }
    }
}
```

```
        const slotNumber = id.toString();
        this.memorySlots[fileName][slotNumber] =
combination;
    }
    else if (generateType === 'multiFile') {
        // Each combination gets its own file
        const fileNumber = id.toString();
        if (!this.memorySlots[fileNumber]) {
            this.memorySlots[fileNumber] = {};
        }
        // Always use slot 1 for each file
        this.memorySlots[fileNumber]['1'] =
combination;
    }

    // Add to file content for download
    if (generateType === 'singleFile') {
        fileContent += `SLOT
${id}\n${combination}\n-----\n`;
    } else {
        fileContent += `FILE ${id}\nSLOT
1\n${combination}\n-----\n`;
    }
}

// Update progress approximately every 5%
if (batch % Math.max(1, Math.floor(totalBatches / 20)) === 0 || batch === totalBatches - 1) {
    const progress = ((batch + 1) / totalBatches) *
100;
    progressCallback(progress, `Generated
${Math.min((batch + 1) * batchSize, nbrComb)} of ${nbrComb} combinations
(${progress.toFixed(1)}%)`);
}
}

// Format the complete output in a way that's ready for
the application
let finalContent = `GENERATED COMBINATIONS\n`;
finalContent += `Generation Mode: ${generateType} ===
'singleFile' ? 'Single File' : 'One File Per Combination'\n`;
finalContent += `(k+1)^n = (${k} + 1)^${n} =
${nbrComb}\n`;
finalContent +=
`=====`;
finalContent += fileContent;
finalContent += `\n\nEnd. Total combinations:
${nbrComb}\n`;

return {
    content: finalContent,
    count: nbrComb,
    type: generateType,
    fileName: fileName
}
```

```
        };
    }

    // Import generated combinations from text based on generation
    type
    importCombinationsFromText(text, type, fileName) {
        try {
            if (type === 'singleFile') {
                // Parse single file format
                const pattern = /SLOT (\d+)\n([\s\S]*?)(?=\\-
{20}|\\Z)/g;
                let match;
                let count = 0;

                // Create or clear the target file
                if (!this.memorySlots[fileName]) {
                    this.memorySlots[fileName] = {};
                } else {
                    // Clear existing slots
                    this.memorySlots[fileName] = {};
                }

                while ((match = pattern.exec(text)) !== null) {
                    const slotNumber = match[1];
                    let value = match[2].trim();

                    // Store in memory slots
                    this.memorySlots[fileName][slotNumber] =
                        value;
                    count++;
                }
            }

            return `Imported ${count} combinations into file
"${fileName}"`;
        }
        else if (type === 'multiFile') {
            // Parse multi-file format
            const filePattern = /FILE (\d+)\nSLOT
(\d+)\n([\s\S]*?)(?=\\-{20}|\\Z)/g;
            let fileMatch;
            let count = 0;

            while ((fileMatch = filePattern.exec(text)) !==
null) {
                const lineNumber = fileMatch[1];
                const slotNumber = fileMatch[2];
                let value = fileMatch[3].trim();

                // Create file if it doesn't exist
                if (!this.memorySlots[lineNumber]) {
                    this.memorySlots[lineNumber] = {};
                }

                // Store in memory slots
            }
        }
    }
}
```

```
        this.memorySlots[fileNumber][slotNumber] =
value;
        count++;
    }

    // Fallback for standard format without FILE
headers
    if (count === 0) {
        const pattern = /SLOT (\d+)\n([\s\S]*?)\n=\\-{20}|\\Z/g;
        let match;
        let fileNumber = 1;

        while ((match = pattern.exec(text)) !== null)
{
            const slotNumber = match[1];
            let value = match[2].trim();

            // Create file if it doesn't exist
            if
(!this.memorySlots[fileNumber.toString()]) {

this.memorySlots[fileNumber.toString()] = {};
}

            // Store in memory slots
            this.memorySlots[fileNumber.toString()]
['1'] = value;
            count++;
            fileNumber++;
        }
    }

    return `Imported ${count} combinations into
separate files`;
}

    return "Unknown generation type for import";
} catch (e) {
    return `Error importing combinations:
${e.message}`;
}
}

// Clear all memory slots
clearAll() {
    this.memorySlots = {};
    return 'All memory slots cleared';
}
}

// Proof by Contradiction Engine integration
class ProofByContradictionEngine {
constructor() {
```

```
        this.statementsFileDialog = "statements"; // File ID for
storing statements in memory manager
        this.char_set =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ+-*/=<>!@#$%^&()[]"
{},;:.?~\`|";
        this.MAX_STATEMENT_LENGTH = 256;
        this.NUM_CHARACTERS = this.char_set.length;

        // Initialize the statements file
        if (!memoryManager.memorySlots[this.statementsFileDialog]) {
            memoryManager.memorySlots[this.statementsFileDialog] = {};
        }
    }

    clearStatements() {
        memoryManager.memorySlots[this.statementsFileDialog] = {};
        return "All statements cleared.";
    }

    addStatement(statement) {
        // Get the next slot number
        let slotNumber = 1;
        const existingSlots =
Object.keys(memoryManager.memorySlots[this.statementsFileDialog])
    .map(key => parseInt(key))
    .filter(num => !isNaN(num));

        if (existingSlots.length > 0) {
            slotNumber = Math.max(...existingSlots) + 1;
        }

        // Store the statement
        memoryManager.memorySlots[this.statementsFileDialog]
[slotNumber] = statement;
        return slotNumber;
    }

    getStatements() {
        const statements = [];
        const slots =
memoryManager.memorySlots[this.statementsFileDialog];

        const sortedKeys = Object.keys(slots)
    .map(key => parseInt(key))
    .filter(num => !isNaN(num))
    .sort((a, b) => a - b);

        for (const key of sortedKeys) {
            statements.push(slots[key]);
        }

        return statements;
    }
}
```

```
generateStatement() {
    const length = Math.floor(Math.random() *
this.MAX_STATEMENT_LENGTH) + 1;
    let statement = '';

    for (let i = 0; i < length; i++) {
        statement += this.char_set[Math.floor(Math.random() *
this.NUM_CHARACTERS)];
    }

    return statement;
}

generateStatements(numStatements) {
    this.clearStatements();

    for (let i = 0; i < numStatements; i++) {
        const statement = this.generateStatement();
        this.addStatement(statement);
    }

    return `\$${numStatements} statements generated.`;
}

checkContradiction() {
    const statements = this.getStatements();

    // Implementation of rule_statement_not_statement
    for (let i = 0; i < statements.length; i++) {
        const statement = statements[i];
        const notStatement = `NOT \$${statement}\`;

        for (let j = 0; j < statements.length; j++) {
            if (i !== j && statements[j] === notStatement) {
                return {
                    found: true,
                    indices: [i, j],
                    statements: [statement, notStatement]
                };
            }
        }
    }

    // Implementation of rule_identical_statements
    for (let i = 0; i < statements.length; i++) {
        for (let j = i + 1; j < statements.length; j++) {
            if (statements[i] === statements[j]) {
                return {
                    found: true,
                    indices: [i, j],
                    statements: [statements[i],
statements[j]],
                    rule: "identical_statements"
                };
            }
        }
    }
}
```

```
        }
    }

    return { found: false };
}

processInstruction(instruction) {
    const parts = instruction.split(' ');
    const cmd = parts[0];

    switch (cmd) {
        case 'ASSERT':
            if (parts.length < 2) {
                return "Error: ASSERT requires a statement.";
            }
            const statement = parts.slice(1).join(' ');
            const slotNumber = this.addStatement(statement);
            return `\`Added statement "${statement}" at
position ${slotNumber}\`;

        case 'NOT':
            if (parts.length < 2) {
                return "Error: NOT requires a statement.";
            }
            const notStatement = parts.slice(1).join(' ');
            const notSlotNumber = this.addStatement(`NOT
${notStatement}`);
            return `\`Added statement "NOT ${notStatement}" at
position ${notSlotNumber}\`;

        case 'AND':
            if (parts.length < 3) {
                return "Error: AND requires two statements.";
            }
            const andStatement1 = parts[1];
            const andStatement2 = parts.slice(2).join(' ');
            const andSlotNumber =
this.addStatement(`(\`${andStatement1}\`) AND (\`${andStatement2}\`)`);
            return `\`Added statement "(\${andStatement1}) AND
(\${andStatement2})" at position ${andSlotNumber}\`;

        case 'OR':
            if (parts.length < 3) {
                return "Error: OR requires two statements.";
            }
            const orStatement1 = parts[1];
            const orStatement2 = parts.slice(2).join(' ');
            const orSlotNumber =
this.addStatement(`\` ${orStatement1} OR ${orStatement2}\``);
            return `\`Added statement "\${orStatement1} OR
\${orStatement2}" at position ${orSlotNumber}\`;

        case 'IMPLIES':
    }
}
```

```
        if (parts.length < 3) {
            return "Error: IMPLIES requires two
statements.";
        }
        const impliesStatement1 = parts[1];
        const impliesStatement2 = parts.slice(2).join('
');
        const impliesSlotNumber =
this.addStatement(`\` ${impliesStatement1} IMPLIES ${impliesStatement2}\` `);
        return `\` Added statement "${impliesStatement1}
IMPLIES ${impliesStatement2}" at position ${impliesSlotNumber}.`\`;

        case 'CONTRADICTION':
        if (parts.length < 3) {
            return "Error: CONTRADICTION requires two
statements.";
        }

        const contradictionStatement1 = parts[1];
        const contradictionStatement2 =
parts.slice(2).join(' ');

        // Check if the statements exist in our framework
        const statements = this.getStatements();
        let found1 = false, found2 = false;
        let index1 = -1, index2 = -1;

        for (let i = 0; i < statements.length; i++) {
            if (statements[i] === contradictionStatement1)
{
                found1 = true;
                index1 = i;
            }
            if (statements[i] === contradictionStatement2)
{
                found2 = true;
                index2 = i;
            }
        }

        if (found1 && found2) {
            return `\` Contradiction found between
statements ${index1 + 1} and ${index2 + 1}:`\\n  ${contradictionStatement1}\\n
${contradictionStatement2}\` `;
        } else {
            return "No contradiction found between the
specified statements.";
        }

        default:
            return `\` Unknown instruction: ${cmd}\` `;
    }
}
```

```
// Initialize the applications
const memoryManager = new MemorySlotManager();
const logicEngine = new ProofByContradictionEngine();

// DOM Elements
const terminal = document.getElementById('terminal');
const commandInput = document.getElementById('commandInput');
const multilineInput = document.getElementById('multilineInput');
const submitCommandBtn = document.getElementById('submitCommand');
const submitMultilineBtn =
document.getElementById('submitMultiline');
const cancelMultilineBtn =
document.getElementById('cancelMultiline');
const memoryDisplay = document.getElementById('memoryDisplay');
const systemTimeEl = document.getElementById('systemTime');

const logicTerminal = document.getElementById('logicTerminal');
const logicCommandInput =
document.getElementById('logicCommandInput');
const submitLogicCommandBtn =
document.getElementById('submitLogicCommand');
const checkContradictionsBtn =
document.getElementById('checkContradictions');
const clearStatementsBtn =
document.getElementById('clearStatements');
const generateStatementsBtn =
document.getElementById('generateStatements');

const generationModal =
document.getElementById('generationModal');
const generationProgress =
document.getElementById('generationProgress');
const generationStatus =
document.getElementById('generationStatus');
const downloadGeneratedBtn =
document.getElementById('downloadGeneratedBtn');
const importGeneratedBtn =
document.getElementById('importGeneratedBtn');
const startGenerationBtn =
document.getElementById('startGenerationBtn');
const fileNameInput = document.getElementById('fileNameInput');
let generatedResults = null;

// Set current time
const now = new Date();
systemTimeEl.textContent = `SYSTEM INITIALIZED:
${now.toISOString().replace('T', ' ').substr(0, 19)}`;

// Add initial welcome message
appendToTerminal('INTEGRATED MEMORY & LOGIC FRAMEWORK', 'system');
appendToTerminal('Type "help" for a list of commands.', 'system');

// Add initial welcome message to logic terminal
```

```
        appendToLogicTerminal('PROOF BY CONTRADICTION ENGINE INITIALIZED',
'system');

        appendToLogicTerminal('Type "help" for a list of instructions.',
'system');

        // Set up tab navigation
document.querySelectorAll('.nav-tab').forEach(tab => {
    tab.addEventListener('click', function() {
        // Remove active class from all tabs
document.querySelectorAll('.nav-tab').forEach(t => {
    t.classList.remove('active');
});

        // Add active class to clicked tab
this.classList.add('active');

        // Hide all tab content
document.querySelectorAll('.tab-content').forEach(content
=> {
    content.classList.remove('active');
});

        // Show the selected tab content
const tabId = this.getAttribute('data-tab');

document.getElementById(`#${tabId}Tab`).classList.add('active');
});

});

        // Event handler for generate type change

document.querySelectorAll('input[name="generateType"]').forEach(radio => {
    radio.addEventListener('change', function() {
        const fileInputContainer =
document.getElementById('fileInputContainer');
        if (this.value === 'singleFile') {
            fileInputContainer.style.display = 'block';
        } else {
            fileInputContainer.style.display = 'none';
        }
    });
});

        // Start generation button handler
startGenerationBtn.addEventListener('click', async () => {
    const generateType =
document.querySelector('input[name="generateType"]:checked').value;
    let fileName = fileNameInput.value.trim();

        // Default to "combinations" if empty
if (generateType === 'singleFile' && !fileName) {
    fileName = 'combinations';
}
```

```
try {
    startGenerationBtn.disabled = true;
    downloadGeneratedBtn.disabled = true;
    importGeneratedBtn.disabled = true;
    generationProgress.style.width = '0%';
    generationStatus.textContent = 'Starting generation...';

    // Get n value from stored data
    const n = parseInt(generationModal.dataset.n);
    const chars = generationModal.dataset.customChars ||

memoryManager.generateCharSet();

    generatedResults = await
memoryManager.generateCombinations(
    chars,
    n,
    generateType,
    fileName,
    (progress, status) => {
        generationProgress.style.width =
```${progress}%```;
        generationStatus.textContent = status;
    }
);

    downloadGeneratedBtn.disabled = false;
    importGeneratedBtn.disabled = false;

    let modeDesc = generateType === 'singleFile' ?
        `single file "${fileName}"` :
        'separate files (one per combination)';

    generationStatus.textContent = `Generation complete!
${generatedResults.count} combinations generated in ${modeDesc}.`;

    // If the count is manageable, automatically import
    if (generatedResults.count <= 1000) {
        const result =
memoryManager.importCombinationsFromText(
            generatedResults.content,
            generatedResults.type,
            generatedResults.fileName
        );
        appendToTerminal(result, 'system');
        updateMemoryDisplay();
    }
} catch (e) {
    generationStatus.textContent = `Error during generation:
${e.message}`;
    appendToTerminal(`Error during generation:
${e.message}`, 'system');
} finally {
    startGenerationBtn.disabled = false;
}
```

```
});

// Generate statements button handler
generateStatementsBtn.addEventListener('click', () => {

document.getElementById('statementsGenerationModal').style.display = 'block';
});

// Start statements generation button handler

document.getElementById('startStatementsGenBtn').addEventListener('click', () => {
    const numStatements =
parseInt(document.getElementById('numStatementsInput').value);
    if (isNaN(numStatements) || numStatements <= 0) {
        appendToLogicTerminal('Error: Please enter a valid
positive number.', 'system');
        return;
    }

    const result = logicEngine.generateStatements(numStatements);
    appendToLogicTerminal(result, 'system');

document.getElementById('statementsGenerationModal').style.display = 'none';

        // Display the current statements
        const statements = logicEngine.getStatements();
        let output = 'Current statements in framework:';
        for (let i = 0; i < statements.length; i++) {
            output += `\\n${i + 1}. ${statements[i]}`;
        }
        appendToLogicTerminal(output, 'system');
    });

    // Check contradictions button handler
    checkContradictionsBtn.addEventListener('click', () => {
        const statements = logicEngine.getStatements();

        if (statements.length === 0) {
            appendToLogicTerminal('No statements in framework.',
'system');
            return;
        }

        let output = 'Current statements in framework:';
        for (let i = 0; i < statements.length; i++) {
            output += `\\n${i + 1}. ${statements[i]}`;
        }
        appendToLogicTerminal(output, 'system');

        const result = logicEngine.checkContradiction();
        if (result.found) {
            let contradictionMsg = `Contradiction found between
statements ${result.indices[0] + 1} and ${result.indices[1] + 1}:\\n`;
            contradictionMsg += `  ${result.statements[0]}\\n
`;
```

```
\${result.statements[1]}\n`;  
  
        if (result.rule) {  
            contradictionMsg += ` (Detected by \${result.rule}  
rule)`;  
        }  
  
        appendToLogicTerminal(contradictionMsg, 'system');  
    } else {  
        appendToLogicTerminal('No contradictions found in the  
current framework.', 'system');  
    }  
});  
  
// Clear statements button handler  
clearStatementsBtn.addEventListener('click', () => {  
    const result = logicEngine.clearStatements();  
    appendToLogicTerminal(result, 'system');  
});  
  
// Enable tab key in the multiline input  
multilineInput.addEventListener('keydown', function(e) {  
    if (e.key === 'Tab') {  
        e.preventDefault();  
  
        // Insert a tab at the current cursor position  
        const start = this.selectionStart;  
        const end = this.selectionEnd;  
        const value = this.value;  
  
        // Insert the tab character  
        this.value = value.substring(0, start) + '\\t' +  
value.substring(end);  
  
        // Move the cursor after the tab  
        this.selectionStart = this.selectionEnd = start + 1;  
    }  
});  
  
// Event listeners for memory manager  
commandInput.addEventListener('keydown', (e) => {  
    if (e.key === 'Enter') {  
        const command = commandInput.value.trim();  
        submitCommand(command);  
    }  
});  
  
submitCommandBtn.addEventListener('click', () => {  
    const command = commandInput.value.trim();  
    submitCommand(command);  
});  
  
submitMultilineBtn.addEventListener('click', () => {  
    const multilineValue = multilineInput.value;
```

```
        finishMultilineInput(multilineValue);
    });

cancelMultilineBtn.addEventListener('click', () => {
    // Reset the UI
    resetToCommandMode();
    appendToTerminal('Multi-line input canceled.', 'system');
});

// Event listeners for logic engine
logicCommandInput.addEventListener('keydown', (e) => {
    if (e.key === 'Enter') {
        const command = logicCommandInput.value.trim();
        submitLogicCommand(command);
    }
});

submitLogicCommandBtn.addEventListener('click', () => {
    const command = logicCommandInput.value.trim();
    submitLogicCommand(command);
});

downloadGeneratedBtn.addEventListener('click', () => {
    if (generatedResults) {
        const blob = new Blob([generatedResults.content], { type: 'text/plain' });
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = 'GENERATED_COMBINATIONS.txt';
        a.click();
        URL.revokeObjectURL(url);
    }
});

importGeneratedBtn.addEventListener('click', () => {
    if (generatedResults) {
        // Import the generated combinations into memory slots
        const result = memoryManager.importCombinationsFromText(
            generatedResults.content,
            generatedResults.type,
            generatedResults.fileName
        );
        appendToTerminal(result, 'system');
        updateMemoryDisplay();
        document.getElementById('generationModal').style.display =
'nave';
    }
});

// Function to append text to the terminal
function appendToTerminal(text, type = 'command') {
    const entryDiv = document.createElement('div');
```

```
if (type === 'input') {
    entryDiv.className = 'terminal-input';
    entryDiv.innerHTML = `<span></span>
${escapeHtml(text)}</span>`;
} else if (type === 'system') {
    entryDiv.className = 'terminal-system';
    entryDiv.textContent = text;
} else {
    entryDiv.className = 'terminal-output';
    entryDiv.innerHTML = text.replace(/\n/g, '<br>');
}

terminal.appendChild(entryDiv);
terminal.scrollTop = terminal.scrollHeight;
}

// Function to append text to the logic terminal
function appendToLogicTerminal(text, type = 'command') {
    const entryDiv = document.createElement('div');

    if (type === 'input') {
        entryDiv.className = 'terminal-input';
        entryDiv.innerHTML = `<span></span>
${escapeHtml(text)}</span>`;
    } else if (type === 'system') {
        entryDiv.className = 'terminal-system';
        entryDiv.textContent = text;
    } else {
        entryDiv.className = 'terminal-output';
        entryDiv.innerHTML = text.replace(/\n/g, '<br>');
    }

    logicTerminal.appendChild(entryDiv);
    logicTerminal.scrollTop = logicTerminal.scrollHeight;
}

function escapeHtml(text) {
    const div = document.createElement('div');
    div.textContent = text;
    return div.innerHTML;
}

// Clear the terminal
function clearTerminal() {
    terminal.innerHTML = '';
    appendToTerminal('Terminal cleared.', 'system');
}

// Clear the logic terminal
function clearLogicTerminal() {
    logicTerminal.innerHTML = '';
    appendToLogicTerminal('Terminal cleared.', 'system');
}
```

```
// Process a command for memory manager
async function submitCommand(command) {
    if (!command) return;

    appendToTerminal(command, 'input');
    commandInput.value = '';

    const commandParts = command.split(' ');
    const cmd = commandParts[0].toLowerCase();

    try {
        switch (cmd) {
            case 'help':
            case '?':
                document.getElementById('helpModal').style.display
= 'block';
                break;

            case 'assign':
                if (commandParts.length < 3) {
                    appendToTerminal('Error: Missing arguments for
assign command.', 'system');
                    break;
                }

                const fileName = commandParts[1];
                const slotNumber = commandParts[2];

                // Switch to multi-line input mode
                switchToMultilineMode('assign', fileName,
slotNumber);
                break;

            case 'read':
                if (commandParts.length < 3) {
                    appendToTerminal('Error: Missing arguments for
read command.', 'system');
                    break;
                }

                const readResult =
memoryManager.readSlot(commandParts[1], commandParts[2]);
                appendToTerminal(readResult, 'system');
                break;

            case 'last_slot':
                if (commandParts.length < 2) {
                    appendToTerminal('Error: Missing file number
for last_slot command.', 'system');
                    break;
                }

                const lastSlot =
memoryManager.getLastSlotNumber(commandParts[1]);
```

```
        if (lastSlot !== null) {
            appendToTerminal(`Last slot number in file
\${commandParts[1]} is \${lastSlot}.`, 'system');
        } else {
            appendToTerminal(`No slots found in file
\${commandParts[1]}.`, 'system');
        }
        break;

    case 'search':
        // Switch to multi-line input mode
        switchToMultilineMode('search');
        break;

    case 'call':
        if (commandParts.length < 4) {
            appendToTerminal('Error: Missing arguments for
call command.', 'system');
            break;
        }

        try {
            const callFileNumber = commandParts[1];
            const startSlot = parseInt(commandParts[2]);
            const endSlot = parseInt(commandParts[3]);
            const callResult =
memoryManager.callSlotRange(callFileNumber, startSlot, endSlot);
            appendToTerminal(callResult, 'system');
        } catch (e) {
            appendToTerminal('Error: Slot numbers must be
integers.', 'system');
        }
        break;

    case 'export':
        if (commandParts.length < 2) {
            appendToTerminal('Error: Missing file number
for export command.', 'system');
            break;
        }

        const exportResult =
memoryManager.exportToTextFile(commandParts[1]);
        appendToTerminal(exportResult, 'system');
        break;

    case 'generate':
        if (commandParts.length < 2) {
            appendToTerminal('Error: Please specify the
number of cells (n).', 'system');
            break;
        }

        const n = parseInt(commandParts[1]);
```

```
        if (n <= 0) {
            appendToTerminal('Error: n must be a positive
integer.', 'system');
            break;
        }

        if (n > 4) {
            appendToTerminal('Warning: Large values of n
will generate very large outputs.', 'system');
            if (!confirm('Continue with generating a large
number of combinations? n=' + n)) {
                appendToTerminal('Generation canceled.',
'system');
                break;
            }
        }

        const charSet = memoryManager.generateCharSet();
        appendToTerminal(`Using standard character set
(${charSet.length} chars)\`, 'system');

        // Show the generation modal with options
        generationModal.style.display = 'block';
        generationProgress.style.width = '0%';
        generationStatus.textContent = 'Select options and
click "Start Generation";

        // Set up initial state for the modal

document.querySelector('#generateSingleFile').checked = true;

document.getElementById('fileInputContainer').style.display = 'block';
    fileNameInput.value = 'combinations';
    downloadGeneratedBtn.disabled = true;
    importGeneratedBtn.disabled = true;
    startGenerationBtn.disabled = false;

        // Store n value for later use
        generationModal.dataset.n = n;
        generationModal.dataset.customChars = '';

        break;

        case 'custom_generate':
            if (commandParts.length < 2) {
                appendToTerminal('Error: Please specify the
number of cells (n).', 'system');
                break;
            }

            const customN = parseInt(commandParts[1]);
            if (customN <= 0) {
                appendToTerminal('Error: n must be a positive
integer.', 'system');
```

```
        break;
    }

    // Switch to multi-line input mode for custom
character set
    switchToMultilineMode('custom_generate', customN);
    break;

    case 'display':
    case 'show':
        updateMemoryDisplay();
        appendToTerminal('Memory display updated.',
'system');
        break;

    case 'clear':
        clearTerminal();
        break;

    case 'save_json':
        const saveFileNumber = commandParts.length > 1 ?
commandParts[1] : null;
        const saveResult =
memoryManager.saveToJson(saveFileNumber);
        appendToTerminal(saveResult, 'system');
        break;

    case 'save_all':
        const saveAllResult =
memoryManager.saveAllAsZip();
        appendToTerminal(saveAllResult, 'system');
        break;

    case 'load_json':
        // Create a file input element
        const fileInput = document.createElement('input');
        fileInput.type = 'file';
        fileInput.accept = '.json';
        fileInput.style.display = 'none';
        document.body.appendChild(fileInput);

        fileInput.onchange = function(e) {
            const file = e.target.files[0];
            if (!file) {
                appendToTerminal('No file selected.',
'system');
                return;
            }

            memoryManager.loadFromFile(file)
            .then(result => {
                appendToTerminal(result, 'system');
                updateMemoryDisplay();
            })
        }
    }
}
```

```
        .catch(error => {
            appendToTerminal(`Error: \${error}\`, 'system');
        });

        inputFile.click();
        document.body.removeChild(inputFile);
        break;

    case 'load_all':
        // Simplified version just using JSON
        const zipInput = document.createElement('input');
        zipInput.type = 'file';
        zipInput.accept = '.json';
        zipInput.style.display = 'none';
        document.body.appendChild(zipInput);

        zipInput.onchange = function(e) {
            const file = e.target.files[0];
            if (!file) {
                appendToTerminal('No file selected.', 'system');
                return;
            }

            memoryManager.loadFromFile(file)
                .then(result => {
                    appendToTerminal(result, 'system');
                    updateMemoryDisplay();
                })
                .catch(error => {
                    appendToTerminal(`Error: \${error}\`, 'system');
                });
        };
    }

    zipInput.click();
    document.body.removeChild(zipInput);
    break;

    case 'exit':
    case 'quit':
        appendToTerminal('This is a web application. To exit, close the browser tab.', 'system');
        break;

    default:
        appendToTerminal(`Unknown command: \${cmd}\`, 'system');
        appendToTerminal("Type 'help' for available commands.", 'system');
    }
} catch (error) {
```

```
        appendToTerminal(`Error executing command:  
${error.message}\`, 'system');  
    }  
}  
  
// Process a command for logic engine  
function submitLogicCommand(command) {  
    if (!command) return;  
  
    appendToLogicTerminal(command, 'input');  
    logicCommandInput.value = '';  
  
    if (command.toLowerCase() === 'help') {  
        document.getElementById('logicHelpModal').style.display =  
'block';  
        return;  
    }  
  
    try {  
        const result = logicEngine.processInstruction(command);  
        appendToLogicTerminal(result, 'system');  
    } catch (error) {  
        appendToLogicTerminal(`Error: ${error.message}\`,  
'system');  
    }  
}  
  
// Switch to multi-line input mode  
function switchToMultilineMode(mode, ...args) {  
    // Hide command input and button  
    commandInput.style.display = 'none';  
    submitCommandBtn.style.display = 'none';  
  
    // Show multi-line input, submit and cancel buttons  
    multilineInput.style.display = 'block';  
    submitMultilineBtn.style.display = 'inline-block';  
    cancelMultilineBtn.style.display = 'inline-block';  
  
    // Clear the multi-line input  
    multilineInput.value = '';  
  
    // Focus the multi-line input  
    multilineInput.focus();  
  
    // Store the mode and arguments  
    multilineInput.dataset.mode = mode;  
    multilineInput.dataset.args = JSON.stringify(args);  
  
    // Set appropriate placeholder text  
    if (mode === 'assign') {  
        multilineInput.placeholder = `Enter value for file  
${args[0]}, slot ${args[1]} (TAB key supported)\`;  
    } else if (mode === 'search') {  
        multilineInput.placeholder = 'Enter search value (TAB key
```

```
supported)';
        } else if (mode === 'custom_generate') {
            multilineInput.placeholder = 'Enter custom character set
for combination generation';
        }

        appendToTerminal(`Enter multi-line value for "\${mode}"
(Submit when finished):\` , 'system');
    }

    // Finish multi-line input
    async function finishMultilineInput(value) {
        const mode = multilineInput.dataset.mode;
        const args = JSON.parse(multilineInput.dataset.args || '[]');

        resetToCommandMode();

        if (mode === 'assign') {
            const [fileNumber, slotNumber] = args;
            const result = memoryManager.assignSlot(fileNumber,
slotNumber, value);
            appendToTerminal(result, 'system');
            updateMemoryDisplay();
        } else if (mode === 'search') {
            if (!value.trim()) {
                appendToTerminal('Error: No search value provided.',

'system');
                return;
            }

            const searchResults = memoryManager.searchValue(value);
            if (searchResults.length > 0) {
                let resultOutput = 'Search Results:';
                for (const result of searchResults) {
                    resultOutput += `\\nFile: \${result.file}, Slot:
\${result.slot}, Value:\n\${result.value}\\n`;
                }
                appendToTerminal(resultOutput, 'system');
            } else {
                appendToTerminal('No results found.', 'system');
            }
        } else if (mode === 'custom_generate') {
            const customN = args[0];
            const customCharSet = value.trim() ||
memoryManager.generateCharSet();

            appendToTerminal(`Using \${value.trim()} ? 'custom' :
'default' character set (\${customCharSet.length} chars)\` , 'system');

            // Show the generation modal with options
            generationModal.style.display = 'block';
            generationProgress.style.width = '0%';
            generationStatus.textContent = 'Select options and click
"Start Generation"';
        }
    }
}
```

```
// Set up initial state for the modal
document.querySelector('#generateSingleFile').checked =
true;

document.getElementById('fileInputContainer').style.display = 'block';
fileNameInput.value = 'combinations';
downloadGeneratedBtn.disabled = true;
importGeneratedBtn.disabled = true;
startGenerationBtn.disabled = false;

// Store data for later use
generationModal.dataset.n = customN;
generationModal.dataset.customChars = customCharSet;
}

}

// Reset to command mode
function resetToCommandMode() {
    multilineInput.style.display = 'none';
    submitMultilineBtn.style.display = 'none';
    cancelMultilineBtn.style.display = 'none';

    commandInput.style.display = 'block';
    submitCommandBtn.style.display = 'inline-block';

    commandInput.focus();
}

// Update the memory display
function updateMemoryDisplay() {
    const memorySlots = memoryManager.memorySlots;

    if (Object.keys(memorySlots).length === 0) {
        memoryDisplay.innerHTML = `

            <div style="text-align: center; color: #888; margin-top: 20px;">

                No memory slots available.
                <br><br>
                Use the terminal to add memory slots.
            </div>

        `;
        return;
    }
}

let html = '';

// Sort file numbers numerically if possible
const sortedFiles = Object.keys(memorySlots).sort((a, b) => {
    if (/^\d+$/.test(a) && /^\d+$/.test(b)) {
        return parseInt(a) - parseInt(b);
    }
    return a.localeCompare(b);
});
```

```
for (const fileNumber of sortedFiles) {
    const slots = memorySlots[fileNumber];
    const slotCount = Object.keys(slots).length;

        // Skip displaying the statements file in memory display -
it's shown in the logic terminal
        if (fileNumber === logicEngine.statements fileId) {
            continue;
        }

        html += `<div class="memory-file">`;
        html += `<div class="memory-file-title">
FILE ${fileNumber} (${slotCount} slots)
<div class="memory-file-actions">
    <button class="file-action"
onclick="exportFile('${fileNumber}')">Export</button>
    <button class="file-action"
onclick="saveFileAsJson('${fileNumber}')">Save JSON</button>
</div>
</div>
`;

        // Sort slot numbers numerically if possible
        const sortedSlots = Object.keys(slots).sort((a, b) => {
            if (/^\d+$/.test(a) && /^\d+$/.test(b)) {
                return parseInt(a) - parseInt(b);
            }
            return a.localeCompare(b);
        });

        // Limit display to max 10 slots per file for readability
        const displaySlots = sortedSlots.slice(0, 10);
        const remainingSlots = sortedSlots.length - 10;

        for (const slotNumber of displaySlots) {
            const value = slots[slotNumber];

                // Truncate long values for display
            let displayValue = value;
            if (displayValue.length > 100) {
                displayValue = displayValue.substring(0, 97) +
'...';
            }

            html += `<div class="memory-slot">`;
            html += `<div class="memory-slot-title">Slot
${slotNumber}</div>`;
            html += `<div class="memory-slot-
value">${escapeHtml(displayValue)}</div>`;
            html += `</div>`;
        }
}
```

```
        if (remainingSlots > 0) {
            html += `<div style="font-style: italic; margin-top:
10px; text-align: center;">
                ...
            ... and ${remainingSlots} more slots (not
            displayed)
            </div>`;
        }

        html += `</div>`;
    }

    if (html === '') {
        memoryDisplay.innerHTML = `

            <div style="text-align: center; color: #888; margin-
            top: 20px;">
                No memory slots available.
                <br><br>
                Use the terminal to add memory slots.
            </div>
        `;
    } else {
        memoryDisplay.innerHTML = html;
    }
}

// Export a file
function exportFile(fileNumber) {
    const result = memoryManager.exportToTextFile(fileNumber);
    appendToTerminal(result, 'system');
}

// Save a file as JSON
function saveFileAsJson(fileNumber) {
    const result = memoryManager.saveToJson(fileNumber);
    appendToTerminal(result, 'system');
}

// Save all memory slots to a file
function saveAllMemorySlots() {
    const result = memoryManager.saveAllAsZip();
    appendToTerminal(result, 'system');
}

// Load memory slots from a file
async function loadMemorySlotsFromFile() {
    // Create a file input element
    const fileInput = document.createElement('input');
    fileInput.type = 'file';
    fileInput.accept = '.json';
    fileInput.style.display = 'none';
    document.body.appendChild(fileInput);

    fileInput.onchange = function(e) {
        const file = e.target.files[0];
        ...
    }
}
```

```
        if (!file) {
            appendToTerminal('No file selected.', 'system');
            return;
        }

        memoryManager.loadFromFile(file)
            .then(result => {
                appendToTerminal(result, 'system');
                updateMemoryDisplay();
            })
            .catch(error => {
                appendToTerminal(`Error: ${error}`, 'system');
            });
    };

    fileInput.click();
    document.body.removeChild(fileInput);
}

// Clear all memory slots
function clearAllMemorySlots() {
    if (confirm('Are you sure you want to clear all memory slots?
This cannot be undone.')) {
        const result = memoryManager.clearAll();
        appendToTerminal(result, 'system');
        updateMemoryDisplay();
    }
}

// Insert command to input field
function insertCommand(command) {
    commandInput.value = command;
    commandInput.focus();
}

// Insert logic command to logic input field
function insertLogicCommand(command) {
    logicCommandInput.value = command;
    logicCommandInput.focus();
}

// Make utility functions global for access from HTML
window.exportFile = exportFile;
window.saveFileAsJson = saveFileAsJson;
window.saveAllMemorySlots = saveAllMemorySlots;
window.loadMemorySlotsFromFile = loadMemorySlotsFromFile;
window.clearAllMemorySlots = clearAllMemorySlots;
window.clearTerminal = clearTerminal;
window.clearLogicTerminal = clearLogicTerminal;
window.updateMemoryDisplay = updateMemoryDisplay;
window.insertCommand = insertCommand;
window.insertLogicCommand = insertLogicCommand;
window.submitCommand = submitCommand;
```

```
// Check for parent frame communication
if (window.parent !== window) {
    try {
        // Notify parent we're loaded
        if (window.parent.postMessage) {
            window.parent.postMessage({
                action: 'iframe-loaded',
                data: {
                    message: 'Build System iframe loaded'
                }
            }, '*');
        }
    }

    // Check if parent provided an API
    if (window.parentAPI) {
        console.log('Parent API available');
    }
} catch (e) {
    console.warn('Error communicating with parent frame:', e);
}
}

</script>
</body>
</html>`;
```

}

```
/** * Access the memory manager * @returns {Object|null} Memory manager object */
```

```
function getMemoryManager() {
    return _memoryManager;
}
```

```
/** * Access the logic engine * @returns {Object|null} Logic engine object */
```

```
function getLogicEngine() {
    return _logicEngine;
}
```

```
/** * Access the iframe window * @returns {Window|null} Iframe window object */
```

```
function getIframeWindow() {
    return _iframe ? _iframe.contentWindow : null;
}
```

```
/** * Execute a command in the memory manager * @param {string} command - Command to execute * @returns {Promise<any>} Command result */
```

```
/*
function executeMemoryCommand(command) {
    return executeCommand(command);
}

/**
 * Execute a command in the logic engine
 * @param {string} command - Command to execute
 * @returns {Promise<any>} Command result
 */
function executeLogicCommand(command) {
    return new Promise((resolve, reject) => {
        if (!_iframe || !_iframe.contentWindow) {
            reject(new Error('Iframe not available'));
            return;
        }

        try {
            // Process the command
            const iframeWindow = _iframe.contentWindow;

            if (iframeWindow.submitLogicCommand) {
                // For logic terminal commands
                iframeWindow.submitLogicCommand(command);
                resolve(true);
            } else {
                reject(new Error('Logic command execution not available'));
            }
        } catch (error) {
            reject(error);
        }
    });
}

// Public API
return {
    initialize,
    loadBuildSystem,
    showBuildSection,
    getMemoryManager,
    getLogicEngine,
    getIframeWindow,
    executeMemoryCommand,
    executeLogicCommand,
    onLoad
};
})();

// Automatically initialize when the DOM is ready
document.addEventListener('DOMContentLoaded', function() {
    // Look for the main container
    const mainElement = document.querySelector('main');
    if (mainElement) {
        OperatorFramework.Build.initialize({

```

```
        containerId: mainElement.id || 'main'
    });
    console.log('Build System initialized');
} else {
    console.warn('Main element not found, Build System not automatically
initialized');
}
});
```

## [93] integerDatabase/operator-dropdown.js

- **Bytes:** 17992
- **Type:** text

```
/**  
 * operator-dropdown.js  
 *  
 * This file provides the dropdown menu system for the Operator framework,  
 * allowing seamless integration of different subsystems into the main interface.  
 *  
 * @module OperatorDropdown  
 * @version 1.0.0  
 * @author Claude AI  
 */  
  
// Access the shared namespace for Operator framework  
  
/**  
 * Dropdown system for integrating external systems  
 * @namespace OperatorDropdown  
 */  
OperatorFramework.Dropdown = (function() {  
    'use strict';  
  
    // Private variables  
    let _initialized = false;  
    let _availableSystems = [];  
    let _activeSystem = null;  
    let _navigationElement = null;  
  
    /**  
     * System configuration object format  
     * @typedef {Object} SystemConfig  
     * @property {string} id - Unique identifier for the system  
     * @property {string} name - Display name for the system  
     * @property {string} description - Brief description of the system  
     * @property {Function} loadFunction - Function to call to load the system  
     * @property {boolean} [isLoaded=false] - Whether the system is currently  
     loaded  
     */
```

```
/**  
 * Initialize the dropdown system  
 * @param {string} navElementId - ID of the DOM element to contain the  
dropdown  
 * @returns {boolean} Success status  
*/  
function initialize(navElementId) {  
    if (_initialized) {  
        console.warn('OperatorDropdown is already initialized');  
        return false;  
    }  
  
    // Find the navigation element  
    _navigationElement = document.getElementById(navElementId);  
    if (!_navigationElement) {  
        console.error(`Navigation element with ID "${navElementId}" not  
found`);  
        return false;  
    }  
  
    // Add styles for the dropdown  
    _addDropdownStyles();  
  
    // Create the initial dropdown structure  
    _createDropdownStructure();  
  
    _initialized = true;  
    console.log('OperatorDropdown initialized successfully');  
    return true;  
}  
  
/**  
 * Add a system to the dropdown  
 * @param {SystemConfig} systemConfig - The system configuration  
 * @returns {boolean} Success status  
*/  
function addSystem(systemConfig) {  
    if (!_initialized) {  
        console.error('OperatorDropdown not initialized, call initialize()  
first');  
        return false;  
    }  
  
    // Validate the system config  
    if (!_validateSystemConfig(systemConfig)) {  
        return false;  
    }  
  
    // Check for duplicate IDs  
    if (_availableSystems.some(sys => sys.id === systemConfig.id)) {  
        console.error(`System with ID "${systemConfig.id}" already exists`);  
        return false;  
    }  
}
```

```
// Add default properties if not provided
const system = {
  ...systemConfig,
  isLoading: false
};

// Add to available systems
_availableSystems.push(system);

// Add to the dropdown menu
_addSystemToDropdown(system);

console.log(`Added system "${system.name}" to dropdown`);
return true;
}

/**
 * Load a system by ID
 * @param {string} systemId - ID of the system to load
 * @returns {boolean} Success status
 */
function loadSystem(systemId) {
  if (!initialized) {
    console.error('OperatorDropdown not initialized, call initialize() first');
    return false;
  }

  // Find the system
  const system = _availableSystems.find(sys => sys.id === systemId);
  if (!system) {
    console.error(`System with ID "${systemId}" not found`);
    return false;
  }

  // Don't reload if already active
  if (_activeSystem === systemId && system.isLoading) {
    console.log(`System "${system.name}" is already active`);
    return true;
  }

  try {
    // Call the system's load function
    system.loadFunction();

    // Update system status
    system.isLoading = true;
    _activeSystem = systemId;

    // Update UI to indicate active system
    _updateActiveSystemUI(systemId);

    console.log(`Successfully loaded system "${system.name}"`);
    return true;
  }
}
```

```
        } catch (error) {
            console.error(`Error loading system "${system.name}":`, error);
            return false;
        }
    }

/**
 * Unload the currently active system
 * @returns {boolean} Success status
 */
function unloadActiveSystem() {
    if (!_activeSystem) {
        console.log('No active system to unload');
        return false;
    }

    const system = _availableSystems.find(sys => sys.id === _activeSystem);
    if (!system) {
        console.error(`Active system with ID "${_activeSystem}" not found`);
        return false;
    }

    if (typeof system.unloadFunction === 'function') {
        try {
            system.unloadFunction();
        } catch (error) {
            console.error(`Error unloading system "${system.name}":`, error);
        }
    }

    system.isLoaded = false;
    _activeSystem = null;

    // Update UI
    _updateActiveSystemUI(null);

    console.log(`Unloaded system "${system.name}"`);
    return true;
}

/**
 * Get the currently active system
 * @returns {string|null} ID of the active system, or null if none
 */
function getActiveSystem() {
    return _activeSystem;
}

/**
 * Get a list of all available systems
 * @returns {Array<SystemConfig>} Array of system configurations
 */
function getAllSystems() {
    return [..._availableSystems];
```

```
}

/**
 * Check if a system is loaded
 * @param {string} systemId - ID of the system to check
 * @returns {boolean} Whether the system is loaded
 */
function isSystemLoaded(systemId) {
    const system = _availableSystems.find(sys => sys.id === systemId);
    return system ? system.isLoaded : false;
}

/**
 * Validate a system configuration object
 * @private
 * @param {SystemConfig} config - The system configuration to validate
 * @returns {boolean} Whether the configuration is valid
 */
function _validateSystemConfig(config) {
    if (!config) {
        console.error('System configuration is required');
        return false;
    }

    if (!config.id || typeof config.id !== 'string') {
        console.error('System must have a valid string ID');
        return false;
    }

    if (!config.name || typeof config.name !== 'string') {
        console.error('System must have a valid string name');
        return false;
    }

    if (!config.loadFunction || typeof config.loadFunction !== 'function') {
        console.error('System must have a valid load function');
        return false;
    }

    return true;
}

/**
 * Add CSS styles for the dropdown system
 * @private
 */
function _addDropdownStyles() {
    const styleElement = document.createElement('style');
    styleElement.textContent = `
        /* Operator Dropdown Styles */
        .op-dropdown {
            position: relative;
            display: inline-block;
            width: 100%;
```

```
        margin-bottom: 10px;
    }

    .op-dropdown-button {
        width: 100%;
        padding: 10px;
        text-align: center;
        cursor: pointer;
        background-color: var(--primary-color, #4B5320);
        color: white;
        border: 2px solid var(--secondary-color, #BDB76B);
        transition: background-color 0.3s ease;
        font-family: var(--font-main, "Fira Code", "Consolas", monospace);
    }

    .op-dropdown-button:hover {
        background-color: var(--secondary-color, #BDB76B);
        color: var(--text-color, #333333);
    }

    .op-dropdown-content {
        display: none;
        position: absolute;
        background-color: white;
        min-width: 160px;
        box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
        z-index: 1000;
        width: 100%;
    }

    .op-dropdown:hover .op-dropdown-content {
        display: block;
    }

    .op-system-button {
        width: 100%;
        text-align: left;
        padding: 10px;
        display: block;
        background-color: white;
        color: var(--text-color, #333333);
        border: none;
        border-bottom: 1px solid #e0e0e0;
        cursor: pointer;
        transition: background-color 0.2s ease;
        font-family: var(--font-main, "Fira Code", "Consolas", monospace);
    }

    .op-system-button:hover {
        background-color: var(--background-color, #F5F5DC);
    }

    .op-system-button.active {
        border-left: 5px solid var(--primary-color, #4B5320);
    }
```

```
        background-color: var(--background-color, #F5F5DC);
        font-weight: bold;
    }

    .op-system-tooltip {
        position: absolute;
        background-color: #333;
        color: white;
        padding: 5px 10px;
        border-radius: 4px;
        font-size: 0.8rem;
        z-index: 1001;
        visibility: hidden;
        opacity: 0;
        transition: opacity 0.3s;
        max-width: 200px;
        /* Change these properties */
        right: 105%; /* Position to the left instead of using left
property */
        left: auto; /* Clear the left property */
        top: 50%;
        transform: translateY(-50%);
    }

    /* Add a small triangle/arrow pointing right */
    .op-system-tooltip::after {
        content: "";
        position: absolute;
        top: 50%;
        left: 100%; /* Position at the right edge */
        margin-top: -5px;
        border-width: 5px;
        border-style: solid;
        border-color: transparent transparent transparent #333; /* Make
triangle point right */
    }

    .op-system-button:hover .op-system-tooltip {
        visibility: visible;
        opacity: 1;
    }
    ;
    document.head.appendChild(styleElement);
}

/**
 * Create the initial dropdown structure
 * @private
 */
function _createDropdownStructure() {
    const navSection = document.createElement('div');
    navSection.className = 'nav-section';
    navSection.innerHTML =
        <h3>External Systems</h3>
```

```
<div class="op-dropdown">
    <button class="op-dropdown-button">Select System</button>
    <div class="op-dropdown-content" id="op-systems-list">
        <!-- Systems will be added here -->
        <div class="op-no-systems">No systems available</div>
    </div>
</div>
`;

_navigationElement.appendChild(navSection);
}

/** 
 * Add a system to the dropdown UI
 * @private
 * @param {SystemConfig} system - The system to add
 */
function _addSystemToDropdown(system) {
    const systemsList = document.getElementById('op-systems-list');

    // Remove the "no systems" message if it exists
    const noSystems = systemsList.querySelector('.op-no-systems');
    if (noSystems) {
        systemsList.removeChild(noSystems);
    }

    // Create button for the system
    const systemButton = document.createElement('button');
    systemButton.id = `op-system-${system.id}`;
    systemButton.className = 'op-system-button';
    systemButton.dataset.systemId = system.id;
    systemButton.innerHTML =
        `${system.name}
        <span class="op-system-tooltip">${system.description || system.name}</span>
`;
    // Add click handler
    systemButton.addEventListener('click', () => {
        loadSystem(system.id);
    });

    // Add to the dropdown
    systemsList.appendChild(systemButton);
}

/** 
 * Update the UI to indicate the active system
 * @private
 * @param {string|null} systemId - ID of the active system, or null if none
 */
function _updateActiveSystemUI(systemId) {
    // Remove active class from all buttons
    const allButtons = document.querySelectorAll('.op-system-button');
```

```
allButtons.forEach(button => {
    button.classList.remove('active');
});

// Add active class to the active system
if (systemId) {
    const activeButton = document.getElementById(`op-system-${systemId}`);
    if (activeButton) {
        activeButton.classList.add('active');
    }
}

// Update dropdown button text
const dropdownButton = document.querySelector('.op-dropdown-button');
if (dropdownButton) {
    if (systemId) {
        const system = _availableSystems.find(sys => sys.id === systemId);
        dropdownButton.textContent = system ? system.name : 'Select
System';
    } else {
        dropdownButton.textContent = 'Select System';
    }
}

/***
 * Search for systems by name or description
 * @param {string} query - The search query
 * @returns {Array<SystemConfig>} Array of matching system configurations
 */
function searchSystems(query) {
    if (!query || typeof query !== 'string') {
        return [..._availableSystems];
    }

    const normalizedQuery = query.toLowerCase();
    return _availableSystems.filter(system => {
        return system.name.toLowerCase().includes(normalizedQuery) ||
            (system.description &&
            system.description.toLowerCase().includes(normalizedQuery)));
    });
}

/***
 * Filter and update the dropdown based on a search query
 * @param {string} query - The search query
 */
function filterDropdown(query) {
    const matchingSystems = searchSystems(query);
    const systemsList = document.getElementById('op-systems-list');

    // Remove all existing items
    systemsList.innerHTML = '';
}
```

```
if (matchingSystems.length === 0) {
    const noResults = document.createElement('div');
    noResults.className = 'op-no-systems';
    noResults.textContent = 'No matching systems found';
    systemsList.appendChild(noResults);
} else {
    // Add matching systems
    matchingSystems.forEach(system => {
        _addSystemToDropdown(system);
    });

    // Restore active state
    if (_activeSystem) {
        _updateActiveSystemUI(_activeSystem);
    }
}

/**
 * Add a search box to filter systems
 * @param {string} [placeholder="Search systems..."] - Placeholder text for
the search box
 */
function addSearchBox(placeholder = "Search systems...") {
    const navSection = _navigationElement.querySelector('.nav-section:last-child');
    if (!navSection) return;

    const searchContainer = document.createElement('div');
    searchContainer.className = 'op-search-container';
    searchContainer.style.padding = '10px 0';

    const searchInput = document.createElement('input');
    searchInput.type = 'text';
    searchInput.placeholder = placeholder;
    searchInput.className = 'op-search-input';
    searchInput.style.width = '100%';
    searchInput.style.padding = '8px';
    searchInput.style.border = '1px solid var(--secondary-color, #BDB76B)';
    searchInput.style.fontFamily = 'var(--font-main, "Fira Code", "Consolas", monospace)';

    searchInput.addEventListener('input', (e) => {
        filterDropdown(e.target.value);
    });

    searchContainer.appendChild(searchInput);
    navSection.appendChild(searchContainer);
}

// Public API
return {
    initialize,
    addSystem,
```

```
loadSystem,
unloadActiveSystem,
getActiveSystem,
getAllSystems,
isSystemLoaded,
searchSystems,
filterDropdown,
addSearchBox
};

})();

// Automatically initialize when the DOM is ready
document.addEventListener('DOMContentLoaded', function() {
    // Look for the main navigation element
    const mainNav = document.getElementById('main-nav');
    if (mainNav) {
        OperatorFramework.Dropdown.initialize('main-nav');

        // Optional: Add search capability
        OperatorFramework.Dropdown.addSearchBox();
    } else {
        console.warn('Main navigation element #main-nav not found,
OperatorDropdown not initialized');
    }
});
```

## [94] integerDatabase/operator-extensions.js

- **Bytes:** 35813
- **Type:** text

```
/** 
 * Operator Extensions
 * Additional features for the Operator System
 *
 * This file adds advanced features to the quadtree implementation:
 * - Custom cell colors
 * - Custom cell text
 * - Image insertion into cells (file upload and clipboard paste)
 * - Enhanced PNG export
 */

(function() {
    'use strict';

    // Track when the page is fully loaded
    document.addEventListener('DOMContentLoaded', function() {
        // Initialize extensions after a short delay to ensure main app is loaded
        setTimeout(initExtensions, 300);
    });
})();
```

```
/*
 * Initialize extension features
 */
function initExtensions() {
    // Find tree operations section
    const treeSection = document.getElementById('tree-operations');
    if (!treeSection) {
        console.error('Tree operations section not found');
        return;
    }

    // Enhance the quadtree UI
    enhanceQuadtreeUI();

    // Create and add context menu
    createContextMenu();

    // Set up global clipboard paste handler
    setupClipboardPaste();

    // Override the export PNG functionality
    overrideExportPNG();

    console.log('Operator Extensions loaded successfully');
}

/*
 * Enhance the quadtree UI with additional controls
 */
function enhanceQuadtreeUI() {
    const quadtreeContainer = document.getElementById('quadtree-container');
    if (!quadtreeContainer) {
        console.error('Quadtree container not found');
        return;
    }

    // Add right-click event listener to quadtree container
    quadtreeContainer.addEventListener('contextmenu',
        handleQuadtreeContextMenu);

    // Add custom styles
    addCustomStyles();

    // Add file input for images (hidden by default)
    const inputFile = document.createElement('input');
    inputFile.type = 'file';
    inputFile.id = 'quadtree-image-input';
    inputFile.accept = 'image/*';
    inputFile.style.display = 'none';
    document.body.appendChild(inputFile);

    inputFile.addEventListener('change', handleImageUpload);
}
```

```
/**  
 * Add custom styles for the context menu  
 */  
function addCustomStyles() {  
    const styleElement = document.createElement('style');  
    styleElement.textContent = `  
        /* Context Menu Styles */  
        .quadtree-context-menu {  
            position: absolute;  
            background-color: white;  
            border: 1px solid var(--secondary-color);  
            border-radius: 4px;  
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);  
            padding: 8px 0;  
            min-width: 180px;  
            z-index: 1000;  
            display: none;  
        }  
  
        .quadtree-context-menu.active {  
            display: block;  
        }  
  
        .context-menu-item {  
            padding: 8px 16px;  
            cursor: pointer;  
            transition: background-color 0.2s ease;  
            display: flex;  
            align-items: center;  
        }  
  
        .context-menu-item:hover {  
            background-color: var(--background-color);  
        }  
  
        .context-menu-item-icon {  
            margin-right: 10px;  
            width: 16px;  
            text-align: center;  
        }  
  
        .context-menu-separator {  
            height: 1px;  
            background-color: var(--secondary-color);  
            margin: 6px 0;  
            opacity: 0.3;  
        }  
  
        /* Color Picker */  
        .color-picker-container {  
            padding: 8px 16px;  
        }  
  
        /* Cell customizations */`
```

```
.quadtree-cell-custom-text {  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    transform: translate(-50%, -50%);  
    white-space: nowrap;  
    overflow: hidden;  
    text-overflow: ellipsis;  
    max-width: 90%;  
    font-family: var(--font-main);  
    pointer-events: none;  
}  
  
.quadtree-cell-image {  
    position: absolute;  
    top: 0;  
    left: 0;  
    width: 100%;  
    height: 100%;  
    object-fit: cover;  
    pointer-events: none;  
}  
  
/* Hide the original cell content when customized */  
.quadtree-cell.customized .cell-content {  
    display: none;  
}  
  
/* Cell with focus for pasting */  
.quadtree-cell.paste-focus {  
    outline: 2px dashed var(--primary-color);  
    z-index: 10;  
    position: relative;  
}  
  
/* Paste indicator */  
.paste-indicator {  
    position: fixed;  
    bottom: 20px;  
    right: 20px;  
    background-color: var(--background-color);  
    border: 1px solid var(--secondary-color);  
    border-left: 4px solid var(--primary-color);  
    padding: 10px 15px;  
    font-size: 0.9rem;  
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);  
    border-radius: 4px;  
    z-index: 1000;  
    display: none;  
}  
  
.paste-indicator.active {  
    display: block;  
    animation: fadeIn 0.3s ease-in-out;
```

```
}

/* Export loading indicator */
.export-loading {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 2000;
}

.export-loading-content {
    background-color: white;
    padding: 20px;
    border-radius: 4px;
    text-align: center;
}

.export-spinner {
    border: 4px solid #f3f3f3;
    border-top: 4px solid var(--primary-color);
    border-radius: 50%;
    width: 40px;
    height: 40px;
    animation: spin 2s linear infinite;
    margin: 0 auto 15px auto;
}

@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}

/* Custom cell modal */
.quadtree-modal {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5);
    display: flex;
    justify-content: center;
    align-items: center;
```

```
        z-index: 1100;
    }

    .quadtree-modal-content {
        background-color: white;
        padding: 20px;
        border-radius: 4px;
        max-width: 500px;
        width: 100%;
    }

    .quadtree-modal-header {
        display: flex;
        justify-content: space-between;
        align-items: center;
        margin-bottom: 20px;
    }

    .quadtree-modal-close {
        background: none;
        border: none;
        font-size: 1.5rem;
        cursor: pointer;
        padding: 0;
        color: var(--text-color);
    }

    .quadtree-modal-body {
        margin-bottom: 20px;
    }

    .quadtree-form-group {
        margin-bottom: 15px;
    }

    .quadtree-form-group label {
        display: block;
        margin-bottom: 5px;
        font-weight: bold;
    }

    .quadtree-form-group input,
    .quadtree-form-group textarea {
        width: 100%;
        padding: 8px;
        border: 1px solid var(--secondary-color);
        border-radius: 4px;
    }

    .quadtree-form-actions {
        display: flex;
        justify-content: flex-end;
        gap: 10px;
    }
```

```
`;  
    document.head.appendChild(styleElement);  
}  
  
/**  
 * Create the context menu  
 */  
function createContextMenu() {  
    // Check if it already exists  
    if (document.getElementById('quadtree-context-menu')) {  
        return;  
    }  
  
    // Create context menu  
    const contextMenu = document.createElement('div');  
    contextMenu.id = 'quadtree-context-menu';  
    contextMenu.className = 'quadtree-context-menu';  
  
    // Menu items  
    contextMenu.innerHTML = `  
        <div class="context-menu-item" data-action="change-color">  
            <span class="context-menu-item-icon">🎨</span>  
            <span>Change Color...</span>  
        </div>  
        <div class="context-menu-item" data-action="add-text">  
            <span class="context-menu-item-icon">T</span>  
            <span>Add Text...</span>  
        </div>  
        <div class="context-menu-item" data-action="add-image">  
            <span class="context-menu-item-icon">🖼</span>  
            <span>Add Image...</span>  
        </div>  
        <div class="context-menu-item" data-action="paste-image">  
            <span class="context-menu-item-icon">📋</span>  
            <span>Paste From Clipboard</span>  
        </div>  
        <div class="context-menu-separator"></div>  
        <div class="context-menu-item" data-action="reset-cell">  
            <span class="context-menu-item-icon">⟲</span>  
            <span>Reset Cell</span>  
        </div>  
    `;  
  
    // Add context menu to document  
    document.body.appendChild(contextMenu);  
  
    // Add click listeners to menu items  
    contextMenu.querySelectorAll('.context-menu-item').forEach(item => {  
        item.addEventListener('click', handleContextMenuAction);  
    });  
  
    // Click outside to close  
    document.addEventListener('click', function(e) {  
        if (!contextMenu.contains(e.target)) {  
            contextMenu.style.display = 'none';  
        }  
    });  
}
```

```
        contextMenu.classList.remove('active');
    }
});

// Create paste indicator
const pasteIndicator = document.createElement('div');
pasteIndicator.id = 'paste-indicator';
pasteIndicator.className = 'paste-indicator';
pasteIndicator.textContent = 'Press Ctrl+V to paste image';
document.body.appendChild(pasteIndicator);
}

/** 
 * Handle right-click on quadtree cells
 */
function handleQuadtreeContextMenu(event) {
    // Prevent default context menu
    event.preventDefault();

    // Find if click was on a cell
    const cell = event.target.closest('.quadtree-cell');
    if (!cell) return;

    // Store reference to target cell
    window.activeQuadtreeCell = cell;

    // Position and show the context menu
    const contextMenu = document.getElementById('quadtree-context-menu');
    contextMenu.style.left = `${event.clientX}px`;
    contextMenu.style.top = `${event.clientY}px`;
    contextMenu.classList.add('active');
}

/** 
 * Handle context menu actions
 */
function handleContextMenuAction(event) {
    const action = event.currentTarget.getAttribute('data-action');
    const cell = window.activeQuadtreeCell;

    // Hide context menu
    document.getElementById('quadtree-context-menu').classList.remove('active');

    if (!cell) return;

    // Execute action
    switch (action) {
        case 'change-color':
            showColorPickerModal(cell);
            break;

        case 'add-text':
            showTextModal(cell);
    }
}
```

```
        break;

    case 'add-image':
        const fileInput = document.getElementById('quadtree-image-input');
        fileInput.click();
        break;

    case 'paste-image':
        prepareCellForPaste(cell);
        break;

    case 'reset-cell':
        resetCell(cell);
        break;
    }

}

/***
 * Mark cell as customized to hide original content
 */
function markCellAsCustomized(cell) {
    cell.classList.add('customized');
}

/***
 * Show color picker modal
 */
function showColorPickerModal(cell) {
    // Create modal
    const modal = document.createElement('div');
    modal.className = 'quadtree-modal';

    // Modal content
    modal.innerHTML = `
        <div class="quadtree-modal-content">
            <div class="quadtree-modal-header">
                <h3>Change Cell Color</h3>
                <button class="quadtree-modal-close">&times;</button>
            </div>
            <div class="quadtree-modal-body">
                <div class="quadtree-form-group">
                    <label for="cell-color">Select Color:</label>
                    <input type="color" id="cell-color"
value="${getComputedStyle(cell).backgroundColor || '#f0f0f0'}">
                </div>
                <div class="quadtree-form-group">
                    <label for="cell-opacity">Opacity:</label>
                    <input type="range" id="cell-opacity" min="0" max="100"
value="100">
                </div>
            </div>
            <div class="quadtree-form-actions">
                <button class="button button-secondary modal-
cancel">Cancel</button>
            </div>
        </div>
    `;
}
```

```
        <button class="button modal-apply">Apply</button>
    </div>
</div>
`;

// Add modal to document
document.body.appendChild(modal);

// Handle close button
modal.querySelector('.quadtree-modal-close').addEventListener('click', () => {
    document.body.removeChild(modal);
});

// Handle cancel button
modal.querySelector('.modal-cancel').addEventListener('click', () => {
    document.body.removeChild(modal);
});

// Handle apply button
modal.querySelector('.modal-apply').addEventListener('click', () => {
    const color = document.getElementById('cell-color').value;
    const opacity = document.getElementById('cell-opacity').value;

    // Apply color to cell
    cell.style.backgroundColor = color;
    cell.style.opacity = opacity / 100;

    // Mark cell as customized
    markCellAsCustomized(cell);
});

// Close modal
document.body.removeChild(modal);
});

}

/**
 * Show text input modal
 */
function showTextModal(cell) {
    // Find existing text element
    let existingText = '';
    const textElement = cell.querySelector('.quadtree-cell-custom-text');
    if (textElement) {
        existingText = textElement.textContent;
    }

    // Create modal
    const modal = document.createElement('div');
    modal.className = 'quadtree-modal';

    // Modal content
    modal.innerHTML =
        <div class="quadtree-modal-content">
```

```
<div class="quadtree-modal-header">
    <h3>Add Text to Cell</h3>
    <button class="quadtree-modal-close">&times;</button>
</div>
<div class="quadtree-modal-body">
    <div class="quadtree-form-group">
        <label for="cell-text">Text:</label>
        <input type="text" id="cell-text" value="${existingText}">
    </div>
    <div class="quadtree-form-group">
        <label for="text-color">Text Color:</label>
        <input type="color" id="text-color" value="#000000">
    </div>
    <div class="quadtree-form-group">
        <label for="text-size">Text Size:</label>
        <input type="range" id="text-size" min="10" max="32"
value="14">
            <span id="text-size-value">14px</span>
        </div>
    </div>
    <div class="quadtree-form-actions">
        <button class="button button-secondary modal-
cancel">Cancel</button>
        <button class="button modal-apply">Apply</button>
    </div>
</div>
`;

// Add modal to document
document.body.appendChild(modal);

// Update size value display
const sizeInput = document.getElementById('text-size');
const sizeValue = document.getElementById('text-size-value');
sizeInput.addEventListener('input', () => {
    sizeValue.textContent = `${sizeInput.value}px`;
});

// Handle close button
modal.querySelector('.quadtree-modal-close').addEventListener('click', () => {
    document.body.removeChild(modal);
});

// Handle cancel button
modal.querySelector('.modal-cancel').addEventListener('click', () => {
    document.body.removeChild(modal);
});

// Handle apply button
modal.querySelector('.modal-apply').addEventListener('click', () => {
    const text = document.getElementById('cell-text').value;
    const color = document.getElementById('text-color').value;
    const size = document.getElementById('text-size').value;
```

```
// Remove any existing text element
if (textElement) {
    cell.removeChild(textElement);
}

// If text is not empty, add it to the cell
if (text.trim() !== '') {
    const newTextElement = document.createElement('div');
    newTextElement.className = 'quadtree-cell-custom-text';
    newTextElement.textContent = text;
    newTextElement.style.color = color;
    newTextElement.style.fontSize = `${size}px`;

    cell.appendChild(newTextElement);

    // Mark cell as customized
    markCellAsCustomized(cell);
}

// Close modal
document.body.removeChild(modal);
});

}

/** 
 * Handle image upload
 */
function handleImageUpload(event) {
    const cell = window.activeQuadtreeCell;
    if (!cell) return;

    const file = event.target.files[0];
    if (!file) return;

    // Check if file is an image
    if (!file.type.startsWith('image/')) {
        alert('Please select an image file');
        return;
    }

    // Read file as data URL
    const reader = new FileReader();
    reader.onload = function(e) {
        addImageToCell(cell, e.target.result);
    };

    reader.readAsDataURL(file);

    // Reset file input
    event.target.value = '';
}

/**
```

```
* Add an image to a cell
*/
function addImageToCell(cell, imageDataUrl) {
    // Remove any existing image
    const existingImage = cell.querySelector('.quadtree-cell-image');
    if (existingImage) {
        cell.removeChild(existingImage);
    }

    // Create image element
    const img = document.createElement('img');
    img.className = 'quadtree-cell-image';
    img.src = imageDataUrl;

    // Add image to cell
    cell.appendChild(img);

    // Mark cell as customized
    markCellAsCustomized(cell);
}

/**
 * Reset cell to default state
 */
function resetCell(cell) {
    // Remove custom text
    const textElement = cell.querySelector('.quadtree-cell-custom-text');
    if (textElement) {
        cell.removeChild(textElement);
    }

    // Remove image
    const imageElement = cell.querySelector('.quadtree-cell-image');
    if (imageElement) {
        cell.removeChild(imageElement);
    }

    // Reset styles
    cell.style.backgroundColor = '';
    cell.style.opacity = '';

    // Remove customized class to show original content
    cell.classList.remove('customized');

    // Remove paste focus if present
    cell.classList.remove('paste-focus');

    // Trigger a redraw of the cell
    setTimeout(() => {
        const event = new Event('cellreset');
        cell.dispatchEvent(event);
    }, 0);
}
```

```
/**  
 * Prepare a cell for clipboard paste  
 */  
function prepareCellForPaste(cell) {  
    // Remove focus from any previously focused cell  
    const focusedCell = document.querySelector('.quadtree-cell.paste-focus');  
    if (focusedCell) {  
        focusedCell.classList.remove('paste-focus');  
    }  
  
    // Add focus to this cell  
    cell.classList.add('paste-focus');  
  
    // Show the paste indicator  
    const pasteIndicator = document.getElementById('paste-indicator');  
    pasteIndicator.classList.add('active');  
  
    // Auto hide the paste indicator after 3 seconds  
    setTimeout(() => {  
        pasteIndicator.classList.remove('active');  
    }, 3000);  
}  
  
/**  
 * Setup clipboard paste functionality  
 */  
function setupClipboardPaste() {  
    // Listen for paste events on the document  
    document.addEventListener('paste', function(event) {  
        // Find the focused cell for paste  
        const cell = document.querySelector('.quadtree-cell.paste-focus');  
        if (!cell) return;  
  
        // Get clipboard items  
        const items = (event.clipboardData ||  
event.originalEvent.clipboardData).items;  
  
        // Look for image content  
        for (const item of items) {  
            if (item.type.indexOf('image') === 0) {  
                // Get image from clipboard  
                const blob = item.getAsFile();  
                const reader = new FileReader();  
  
                reader.onload = function(e) {  
                    // Add the image to the cell  
                    addImageToCell(cell, e.target.result);  
  
                    // Remove the paste focus  
                    cell.classList.remove('paste-focus');  
  
                    // Hide the paste indicator  
                    document.getElementById('paste-  
indicator').classList.remove('active');  
                };  
                reader.readAsDataURL(blob);  
            }  
        }  
    });  
}
```

```
};

    reader.readAsDataURL(blob);

    // Prevent default paste behavior
    event.preventDefault();
    return;
}
}

// If we get here, no image was found in clipboard
console.log('No image found in clipboard');
});

}

/** 
 * Override the export PNG functionality to include custom elements
 */
function overrideExportPNG() {
    // Find the export button
    const exportButton = document.getElementById('export-quadtree');
    if (!exportButton) {
        console.error('Export button not found');
        return;
    }

    // Store the original click handler
    const originalHandler = exportButton.onclick;

    // Override with our enhanced version
    exportButton.onclick = function(event) {
        // Prevent the default action
        event.preventDefault();

        // Call our enhanced export function
        exportQuadtreeAsPNG();
    };
}

/** 
 * Export the quadtree as a PNG with all custom elements
 */
function exportQuadtreeAsPNG() {
    // Show loading indicator
    showLoadingIndicator('Generating PNG...');

    // Get the quadtree container
    const quadtreeContainer = document.getElementById('quadtree-container');
    if (!quadtreeContainer) {
        hideLoadingIndicator();
        alert('Quadtree container not found');
        return;
    }
}
```

```
// Use html2canvas to capture the DOM elements (including custom elements)
// If html2canvas is not available, we'll use a fallback method
if (typeof html2canvas === 'function') {
    html2canvas(quadtreeContainer, {
        scale: 2, // Higher quality
        backgroundColor: null,
        logging: false,
        useCORS: true, // For images from different origins
        allowTaint: true // Allow cross-origin images
    }).then(canvas => {
        // Convert canvas to PNG and download
        finishExport(canvas);
    }).catch(error => {
        console.error('Error capturing quadtree:', error);
        hideLoadingIndicator();
        alert('Failed to export quadtree: ' + error.message);

        // Try fallback method
        exportQuadtreeWithDOM2Canvas();
    });
} else {
    // Fallback method
    exportQuadtreeWithDOM2Canvas();
}

/**
 * Fallback method to export quadtree using DOM-to-Canvas drawing
 */
function exportQuadtreeWithDOM2Canvas() {
    // Get the quadtree container
    const quadtreeContainer = document.getElementById('quadtree-container');
    if (!quadtreeContainer) {
        hideLoadingIndicator();
        alert('Quadtree container not found');
        return;
    }

    // Create a canvas
    const canvas = document.createElement('canvas');
    const context = canvas.getContext('2d');

    // Set canvas size to match container
    const width = quadtreeContainer.offsetWidth;
    const height = quadtreeContainer.offsetHeight;
    canvas.width = width * 2; // Higher quality
    canvas.height = height * 2;
    context.scale(2, 2);

    // Fill background
    context.fillStyle = 'white';
    context.fillRect(0, 0, width, height);

    // Draw each cell
```

```
const cells = quadtreeContainer.querySelectorAll('.quadtree-cell');

// First, render the cells and their backgrounds
cells.forEach(cell => {
    // Only draw visible cells
    if (cell.style.display !== 'none') {
        // Get cell position and dimensions
        const rect = cell.getBoundingClientRect();
        const containerRect = quadtreeContainer.getBoundingClientRect();

        // Calculate position relative to container
        const x = rect.left - containerRect.left;
        const y = rect.top - containerRect.top;
        const cellWidth = rect.width;
        const cellHeight = rect.height;

        // Draw cell background
        context.fillStyle = getComputedStyle(cell).backgroundColor ||
'#f0f0f0';
        context.globalAlpha = parseFloat(getComputedStyle(cell).opacity) ||
1;
        context.fillRect(x, y, cellWidth, cellHeight);

        // Draw cell border
        context.strokeStyle = getComputedStyle(cell).borderColor ||
'#ccc';
        context.lineWidth = parseFloat(getComputedStyle(cell).borderWidth) ||
1;
        context.globalAlpha = 1;
        context.strokeRect(x, y, cellWidth, cellHeight);
    }
});

// Then, render the content (for proper z-order)
cells.forEach(cell => {
    // Only draw visible cells
    if (cell.style.display !== 'none') {
        // Get cell position and dimensions
        const rect = cell.getBoundingClientRect();
        const containerRect = quadtreeContainer.getBoundingClientRect();

        // Calculate position relative to container
        const x = rect.left - containerRect.left;
        const y = rect.top - containerRect.top;
        const cellWidth = rect.width;
        const cellHeight = rect.height;

        // Draw original cell content if not customized
        if (!cell.classList.contains('customized')) {
            const contentElement = cell.querySelector('.cell-content');
            if (contentElement) {
                context.fillStyle = getComputedStyle(contentElement).color ||
'#333';
                context.font = getComputedStyle(contentElement).font ||

```

```
'12px sans-serif';

        context.textAlign = 'center';
        context.textBaseline = 'middle';
        context.globalAlpha = 1;
        context.fillText(
            contentElement.textContent,
            x + cellWidth / 2,
            y + cellHeight / 2
        );
    }
}

// Draw custom text if present
const textElement = cell.querySelector('.quadtree-cell-custom-
text');
if (textElement) {
    context.fillStyle = getComputedStyle(textElement).color || '#000';
    context.font = getComputedStyle(textElement).font || '14px
sans-serif';
    context.textAlign = 'center';
    context.textBaseline = 'middle';
    context.globalAlpha = 1;
    context.fillText(
        textElement.textContent,
        x + cellWidth / 2,
        y + cellHeight / 2
    );
}

// Draw image if present (this requires handling async loading)
const imageElement = cell.querySelector('.quadtree-cell-image');
if (imageElement) {
    try {
        context.globalAlpha = 1;
        context.drawImage(imageElement, x, y, cellWidth,
cellHeight);
    } catch (error) {
        console.error('Error drawing image:', error);
    }
}
});

// Convert canvas to PNG and download
finishExport(canvas);
}

/**
 * Finish the export process by downloading the canvas as PNG
 */
function finishExport(canvas) {
    try {
        // Convert canvas to data URL

```

```
const dataUrl = canvas.toDataURL('image/png');

// Create download link
const link = document.createElement('a');
link.download = 'quadtree.png';
link.href = dataUrl;
link.click();

// Hide loading indicator
hideLoadingIndicator();
} catch (error) {
    console.error('Error generating PNG:', error);
    hideLoadingIndicator();
    alert('Failed to generate PNG: ' + error.message);
}
}

/**
 * Show a loading indicator
 */
function showLoadingIndicator(message) {
    // Create loading indicator if it doesn't exist
    let loadingEl = document.getElementById('export-loading');
    if (!loadingEl) {
        loadingEl = document.createElement('div');
        loadingEl.id = 'export-loading';
        loadingEl.className = 'export-loading';

        loadingEl.innerHTML =
            `<div class="export-loading-content">
                <div class="export-spinner"></div>
                <div id="export-loading-message">${message || 'Loading...'}</div>
            `;

        document.body.appendChild(loadingEl);
    } else {
        // Update message if it exists
        const messageEl = document.getElementById('export-loading-message');
        if (messageEl) {
            messageEl.textContent = message || 'Loading...';
        }
    }

    // Show the existing element
    loadingEl.style.display = 'flex';
}
}

/**
 * Hide the loading indicator
 */
function hideLoadingIndicator() {
    const loadingEl = document.getElementById('export-loading');
```

```

        if (loadingEl) {
            loadingEl.style.display = 'none';
        }
    }

    /**
     * Load html2canvas library dynamically if needed
     */
    function loadHtml2Canvas() {
        if (typeof html2canvas === 'undefined') {
            return new Promise((resolve, reject) => {
                const script = document.createElement('script');
                script.src =
'https://html2canvas.hertzen.com/dist/html2canvas.min.js';
                script.onload = resolve;
                script.onerror = reject;
                document.head.appendChild(script);
            });
        }

        return Promise.resolve();
    }

    // Initialize html2canvas if not present
    loadHtml2Canvas().catch(error => {
        console.warn('Failed to load html2canvas library:', error);
    });
})();

```

## [95] integerDatabase/operator-framework.js

- **Bytes:** 149
- **Type:** text

```

/**
 * operator-framework.js
 *
 * Base framework that provides the shared namespace for all Operator modules
 */
const OperatorFramework = {};

```

## [96] integerDatabase/operator-language.js

- **Bytes:** 78175
- **Type:** text

```

/**
 * Operator Language System
 * A Turing-complete programming language with graphical interface for the

```

```
Operator System
 */

class OperatorLanguage {
    constructor(canvasId, consoleId) {
        // Language state
        this.variables = new Map(); // Variable store
        this.functions = new Map(); // Function store
        this.callStack = []; // Function call stack
        this.programCounter = 0; // Current instruction pointer
        this.program = []; // Current program
        this.running = false; // Is program running
        this.waitingForInput = false; // Is waiting for input
        this.breakpoints = new Set(); // Set of instruction numbers

        // Runtime configuration
        this.maxIterations = 100000; // Safety against infinite loops
        this.executionDelay = 0; // Delay between statements (ms)
        this.executionMode = 'normal'; // 'normal', 'step', 'fast'

        // Graphics context
        this.canvas = document.getElementById(canvasId);
        this.ctx = this.canvas.getContext('2d');
        this.graphicsStack = []; // For saved states

        // Console output
        this.consoleElement = document.getElementById(consoleId);
        this.consoleHistory = [];
        this.maxConsoleHistory = 100;

        // Initialize standard library
        this.initializeStandardLibrary();
    }

    /**
     * Initialize the standard library of functions
     */
    initializeStandardLibrary() {
        // Math operations
        this.registerNativeFunction('add', (a, b) => a + b);
        this.registerNativeFunction('sub', (a, b) => a - b);
        this.registerNativeFunction('mul', (a, b) => a * b);
        this.registerNativeFunction('div', (a, b) => a / b);
        this.registerNativeFunction('mod', (a, b) => a % b);
        this.registerNativeFunction('pow', (a, b) => Math.pow(a, b));
        this.registerNativeFunction('sqrt', (a) => Math.sqrt(a));
        this.registerNativeFunction('sin', (a) => Math.sin(a));
        this.registerNativeFunction('cos', (a) => Math.cos(a));
        this.registerNativeFunction('tan', (a) => Math.tan(a));
        this.registerNativeFunction('floor', (a) => Math.floor(a));
        this.registerNativeFunction('ceil', (a) => Math.ceil(a));
        this.registerNativeFunction('round', (a) => Math.round(a));
        this.registerNativeFunction('abs', (a) => Math.abs(a));
        this.registerNativeFunction('random', (min, max) => min + Math.random() *
    }
}
```

```
(max - min));

        // Comparison operations
        this.registerNativeFunction('eq', (a, b) => a === b);
        this.registerNativeFunction('neq', (a, b) => a !== b);
        this.registerNativeFunction('lt', (a, b) => a < b);
        this.registerNativeFunction('gt', (a, b) => a > b);
        this.registerNativeFunction('lte', (a, b) => a <= b);
        this.registerNativeFunction('gte', (a, b) => a >= b);
        this.registerNativeFunction('and', (a, b) => a && b);
        this.registerNativeFunction('or', (a, b) => a || b);
        this.registerNativeFunction('not', (a) => !a);

        // String operations
        this.registerNativeFunction('concat', (a, b) => String(a) + String(b));
        this.registerNativeFunction('substr', (str, start, length) =>
String(str).substring(start, start + length));
        this.registerNativeFunction('length', (str) => {
            if (Array.isArray(str)) {
                return str.length;
            } else if (typeof str === 'string') {
                return str.length;
            }
            return 0;
        });
        this.registerNativeFunction('tostring', (a) => String(a));
        this.registerNativeFunction('tonumber', (a) => Number(a));

        // Array operations
        this.registerNativeFunction('array', (...args) => args);
        this.registerNativeFunction('get', (arr, index) => {
            if (!Array.isArray(arr)) {
                throw new Error(`Expected an array, got ${typeof arr}`);
            }
            return arr[index];
        });
        this.registerNativeFunction('set', (arr, index, value) => {
            if (!Array.isArray(arr)) {
                throw new Error(`Expected an array, got ${typeof arr}`);
            }
            const result = [...arr];
            result[index] = value;
            return result;
        });
        this.registerNativeFunction('push', (arr, value) => {
            if (!Array.isArray(arr)) {
                return [value]; // If arr is not an array, create a new array with
the value
            }
            return [...arr, value];
        });
        this.registerNativeFunction('pop', (arr) => {
            if (!Array.isArray(arr) || arr.length === 0) {
                return [];
            }
            return arr.slice(0, -1);
        });
    
```

```
        }
        const result = [...arr];
        result.pop();
        return result;
    });
    this.registerNativeFunction('join', (arr, separator) => {
        if (!Array.isArray(arr)) {
            return String(arr);
        }
        return arr.join(separator || '');
    });
    this.registerNativeFunction('split', (str, separator) => {
        return String(str).split(separator || '');
    });

    // I/O operations
    this.registerNativeFunction('print', (...args) => {
        this.writeToConsole(args.join(' '));
        return args[args.length - 1];
    });
    this.registerNativeFunction('clear', () => {
        this.clearConsole();
        return null;
    });

    // Graphics operations
    this.registerNativeFunction('clearCanvas', (color = '#ffffff') => {
        this.ctx.fillStyle = color;
        this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);
        return null;
    });
    this.registerNativeFunction('setFillColor', (color) => {
        this.ctx.fillStyle = color;
        return color;
    });
    this.registerNativeFunction('setStrokeColor', (color) => {
        this.ctx.strokeStyle = color;
        return color;
    });
    this.registerNativeFunction('setLineWidth', (width) => {
        this.ctx.lineWidth = width;
        return width;
    });
    this.registerNativeFunction('drawRect', (x, y, width, height, fill = true)
=> {
        if (fill) {
            this.ctx.fillRect(x, y, width, height);
        } else {
            this.ctx.strokeRect(x, y, width, height);
        }
        return null;
    });
    this.registerNativeFunction('drawCircle', (x, y, radius, fill = true) => {
        this.ctx.beginPath();
```

```
        this.ctx.arc(x, y, radius, 0, Math.PI * 2);
        if (fill) {
            this.ctx.fill();
        } else {
            this.ctx.stroke();
        }
        return null;
    });
this.registerNativeFunction('drawLine', (x1, y1, x2, y2) => {
    this.ctx.beginPath();
    this.ctx.moveTo(x1, y1);
    this.ctx.lineTo(x2, y2);
    this.ctx.stroke();
    return null;
});
this.registerNativeFunction('drawText', (text, x, y, maxWidth) => {
    if (maxWidth) {
        this.ctx.fillText(text, x, y, maxWidth);
    } else {
        this.ctx.fillText(text, x, y);
    }
    return null;
});
this.registerNativeFunction('setFont', (font) => {
    this.ctx.font = font;
    return font;
});
this.registerNativeFunction('saveGraphicsState', () => {
    this.ctx.save();
    return null;
});
this.registerNativeFunction('restoreGraphicsState', () => {
    this.ctx.restore();
    return null;
});
this.registerNativeFunction('translate', (x, y) => {
    this.ctx.translate(x, y);
    return null;
});
this.registerNativeFunction('rotate', (angle) => {
    this.ctx.rotate(angle);
    return null;
});
this.registerNativeFunction('scale', (x, y) => {
    this.ctx.scale(x, y);
    return null;
});
// Time operations
this.registerNativeFunction('delay', (ms) => {
    return new Promise(resolve => setTimeout(resolve, ms));
});
this.registerNativeFunction('now', () => Date.now());
```

```
// System operations
this.registerNativeFunction('getVariable', (name) =>
this.getVariable(name));
    this.registerNativeFunction('setVariable', (name, value) =>
this.setVariable(name, value));

// Type checking
this.registerNativeFunction('isNumber', (value) => typeof value ===
'number');
    this.registerNativeFunction('isString', (value) => typeof value ===
'string');
    this.registerNativeFunction('isArray', (value) => Array.isArray(value));
    this.registerNativeFunction('isBoolean', (value) => typeof value ===
'boolean');
    this.registerNativeFunction('isNull', (value) => value === null);
    this.registerNativeFunction('isUndefined', (value) => value ===
undefined);
}

/***
 * Register a native JavaScript function in the language
 */
registerNativeFunction(name, func) {
    this.functions.set(name, {
        type: 'native',
        func: func
    });
}

/***
 * Register a user-defined function
 */
defineFunction(name, params, body) {
    this.functions.set(name, {
        type: 'user',
        params: params,
        body: body
    });
}

/***
 * Get a variable value
 */
getVariable(name) {
    if (this.variables.has(name)) {
        return this.variables.get(name);
    }
    return null;
}

/***
 * Set a variable value
 */
setVariable(name, value) {
```

```
        this.variables.set(name, value);
        return value;
    }

    /**
     * Write text to the console
     */
    writeToConsole(text) {
        if (this.consoleElement) {
            const line = document.createElement('div');
            line.className = 'console-line';
            line.textContent = text;
            this.consoleElement.appendChild(line);
            this.consoleElement.scrollTop = this.consoleElement.scrollHeight;

            // Limit console history
            this.consoleHistory.push(text);
            if (this.consoleHistory.length > this.maxConsoleHistory) {
                this.consoleHistory.shift();
                if (this.consoleElement.childNodes.length >
this.maxConsoleHistory) {

                    this.consoleElement.removeChild(this.consoleElement.childNodes[0]);
                }
            }
        }
    }

    /**
     * Clear the console
     */
    clearConsole() {
        if (this.consoleElement) {
            this.consoleElement.innerHTML = '';
            this.consoleHistory = [];
        }
    }

    /**
     * Parse a string into a program
     * This is the main entry point for the parser
     */
    parseProgram(code) {
        const parser = new OperatorLangParser(code, this);
        return parser.parse();
    }

    /**
     * Load a program from a string
     */
    loadProgram(code) {
        this.program = this.parseProgram(code);
        this.programCounter = 0;
        this.running = false;
```

```
        this.callStack = [];
        return this.program;
    }

    /**
     * Set a breakpoint at a specific line
     */
    setBreakpoint(line) {
        this.breakpoints.add(line);
    }

    /**
     * Clear a breakpoint at a specific line
     */
    clearBreakpoint(line) {
        this.breakpoints.delete(line);
    }

    /**
     * Clear all breakpoints
     */
    clearAllBreakpoints() {
        this.breakpoints.clear();
    }

    /**
     * Start program execution
     */
    async run() {
        if (this.running) {
            return;
        }

        this.running = true;
        this.programCounter = 0;
        this.callStack = [];

        return this.executeProgram();
    }

    /**
     * Stop program execution
     */
    stop() {
        this.running = false;
    }

    /**
     * Execute the current program
     */
    async executeProgram() {
        let iterations = 0;

        while (this.running && this.programCounter < this.program.length) {
```

```
// Check for breakpoints
if (this.breakpoints.has(this.programCounter)) {
    this.writeToConsole(`Breakpoint hit at instruction
${this.programCounter}`);
    this.running = false;
    return;
}

// Execute the current instruction
const instruction = this.program[this.programCounter];
this.programCounter++;

try {
    await this.executeInstruction(instruction);
} catch (e) {
    this.writeToConsole(`Runtime error: ${e.message}`);
    this.running = false;
    return;
}

// Safety check against infinite loops
iterations++;
if (iterations > this.maxIterations) {
    this.writeToConsole('Program exceeded maximum iterations - stopping execution');
    this.running = false;
    return;
}

// Add delay between statements if needed
if (this.executionDelay > 0) {
    await new Promise(resolve => setTimeout(resolve,
this.executionDelay));
}

// Step mode - pause after each instruction
if (this.executionMode === 'step') {
    this.running = false;
    return;
}
}

if (this.programCounter >= this.program.length) {
    this.writeToConsole('Program execution completed');
    this.running = false;
}
}

/** 
 * Execute the next step of the program
 */
async step() {
    if (this.programCounter >= this.program.length) {
        this.writeToConsole('Program execution completed');
    }
}
```

```
        return;
    }

    this.running = true;
    const instruction = this.program[this.programCounter];
    this.programCounter++;

    try {
        await this.executeInstruction(instruction);
    } catch (e) {
        this.writeToConsole(`Runtime error: ${e.message}`);
    }

    this.running = false;
}

/**
 * Execute a single instruction
 */
async executeInstruction(instruction) {
    switch (instruction.type) {
        case 'assign':
            const value = await
this.evaluateExpression(instruction.expression);
            this.setVariable(instruction.variable, value);
            return value;

        case 'if':
            const condition = await
this.evaluateExpression(instruction.condition);
            if (condition) {
                // Execute the if body
                for (const stmt of instruction.body) {
                    await this.executeInstruction(stmt);
                    if (!this.running) return;
                }
            } else if (instruction.elseBody) {
                // Execute the else body
                for (const stmt of instruction.elseBody) {
                    await this.executeInstruction(stmt);
                    if (!this.running) return;
                }
            }
            return null;

        case 'while':
            while (this.running) {
                const condition = await
this.evaluateExpression(instruction.condition);
                if (!condition) break;

                for (const stmt of instruction.body) {
                    await this.executeInstruction(stmt);
                    if (!this.running) return;
                }
            }
    }
}
```

```
        }
    }
    return null;

    case 'for':
        const start = await this.evaluateExpression(instruction.start);
        const end = await this.evaluateExpression(instruction.end);

        for (let i = start; i < end && this.running; i++) {
            this.setVariable(instruction.variable, i);

            for (const stmt of instruction.body) {
                await this.executeInstruction(stmt);
                if (!this.running) return;
            }
        }
        return null;

    case 'return':
        const returnValue = await
this.evaluateExpression(instruction.expression);
        return returnValue;

    case 'expression':
        return await this.evaluateExpression(instruction.expression);

    case 'end':
    case 'else':
        // These instruction types should be handled during parsing
        // They should not appear as standalone instructions during
execution
        return null;

    default:
        throw new Error(`Unknown instruction type: ${instruction.type}`);
    }
}

/**
 * Evaluate an expression to a value
 */
async evaluateExpression(expression) {
    switch (expression.type) {
        case 'value':
            return expression.value;

        case 'variable':
            return this.getVariable(expression.name);

        case 'call':
            const func = this.functions.get(expression.function);
            if (!func) {
                throw new Error(`Function not found: ${expression.function}`);
            }
    }
}
```

```
// Evaluate all arguments
const args = [];
for (const arg of expression.arguments) {
    args.push(await this.evaluateExpression(arg));
}

// Call the function
if (func.type === 'native') {
    return await func.func(...args);
} else if (func.type === 'user') {
    // Create new scope for function
    this.callStack.push({
        variables: new Map(this.variables),
        programCounter: this.programCounter
    });

    // Set up parameters
    for (let i = 0; i < func.params.length; i++) {
        this.setVariable(func.params[i], args[i]);
    }

    // Execute function body
    let result = null;
    for (const stmt of func.body) {
        result = await this.executeInstruction(stmt);
        if (!this.running) break;
        if (stmt.type === 'return') break;
    }

    // Restore original scope
    const callFrame = this.callStack.pop();
    this.variables = callFrame.variables;
    this.programCounter = callFrame.programCounter;

    return result;
}
break;

case 'binary':
    const left = await this.evaluateExpression(expression.left);
    const right = await this.evaluateExpression(expression.right);

    switch (expression.operator) {
        case '+': return left + right;
        case '-': return left - right;
        case '*': return left * right;
        case '/': return left / right;
        case '%': return left % right;
        case '==': return left === right;
        case '!=': return left !== right;
        case '<': return left < right;
        case '>': return left > right;
        case '<=': return left <= right;
```

```
        case '>=': return left >= right;
        case '&&': return left && right;
        case '||': return left || right;
        default:
            throw new Error(`Unknown binary operator:
${expression.operator}`);
    }

    case 'unary':
        const value = await
this.evaluateExpression(expression.expression);

        switch (expression.operator) {
            case '!': return !value;
            case '-': return -value;
            default:
                throw new Error(`Unknown unary operator:
${expression.operator}`);
        }

    case 'array':
        const elements = [];
        for (const element of expression.elements) {
            elements.push(await this.evaluateExpression(element));
        }
        return elements;

    case 'object':
        const obj = {};
        for (const key in expression.properties) {
            obj[key] = await
this.evaluateExpression(expression.properties[key]);
        }
        return obj;

    default:
        throw new Error(`Unknown expression type: ${expression.type}`);
}
}

/**
 * Save the program as JSON
 */
exportToJSON() {
    return JSON.stringify({
        program: this.program,
        variables: Array.from(this.variables.entries()),
        functions: Array.from(this.functions.entries())
            .filter(([_, func]) => func.type === 'user')
            .map(([name, func]) => ({
                name,
                params: func.params,
                body: func.body
            })),
    })
}
```

```
        breakpoints: Array.from(this.breakpoints),
        version: '1.0'
    }, null, 2);
}

/** 
 * Load a program from JSON
 */
importFromJSON(json) {
    try {
        const data = JSON.parse(json);

        // Check version
        if (!data.version || data.version !== '1.0') {
            throw new Error('Incompatible program version');
        }

        // Load program
        this.program = data.program;

        // Load variables
        this.variables = new Map(data.variables);

        // Load user-defined functions
        for (const func of data.functions) {
            this.defineFunction(func.name, func.params, func.body);
        }

        // Load breakpoints
        this.breakpoints = new Set(data.breakpoints);

        // Reset state
        this.programCounter = 0;
        this.running = false;
        this.callStack = [];

        return true;
    } catch (e) {
        this.writeToConsole(`Error importing program: ${e.message}`);
        return false;
    }
}

/** 
 * TokenType enum for the lexer
 */
const TokenType = {
    // Literals
    NUMBER: 'NUMBER',
    STRING: 'STRING',
    IDENTIFIER: 'IDENTIFIER',
    TRUE: 'TRUE',
    FALSE: 'FALSE',
```

```
NULL: 'NULL',  
  
// Keywords  
IF: 'IF',  
ELSE: 'ELSE',  
WHILE: 'WHILE',  
FOR: 'FOR',  
IN: 'IN',  
RANGE: 'RANGE',  
FUNC: 'FUNC',  
RETURN: 'RETURN',  
LET: 'LET',  
CONTINUE: 'CONTINUE',  
BREAK: 'BREAK',  
  
// Operators  
PLUS: 'PLUS',  
MINUS: 'MINUS',  
TIMES: 'TIMES',  
DIVIDE: 'DIVIDE',  
MODULO: 'MODULO',  
POWER: 'POWER',  
  
// Comparison operators  
EQUAL: 'EQUAL',  
NOT_EQUAL: 'NOT_EQUAL',  
LESS: 'LESS',  
GREATER: 'GREATER',  
LESS_EQUAL: 'LESS_EQUAL',  
GREATER_EQUAL: 'GREATER_EQUAL',  
  
// Logical operators  
AND: 'AND',  
OR: 'OR',  
NOT: 'NOT',  
  
// Assignment  
ASSIGN: 'ASSIGN',  
  
// Delimiters  
LPAREN: 'LPAREN',  
RPAREN: 'RPAREN',  
LBRACE: 'LBRACE',  
RBRACE: 'RBRACE',  
LBRACKET: 'LBRACKET',  
RBRACKET: 'RBRACKET',  
COMMA: 'COMMA',  
DOT: 'DOT',  
COLON: 'COLON',  
SEMICOLON: 'SEMICOLON',  
  
// End of file  
EOF: 'EOF'  
};
```

```
/*
 * Token class for the lexer
 */
class Token {
    constructor(type, value, line, column) {
        this.type = type;
        this.value = value;
        this.line = line;
        this.column = column;
    }

    toString() {
        return `Token(${this.type}, ${this.value})`;
    }
}

/*
 * Lexer class for tokenizing the input code
 */
class Lexer {
    constructor(source) {
        this.source = source;
        this.position = 0;
        this.line = 1;
        this.column = 1;
        this.tokens = [];

        this.keywords = {
            'if': TokenType.IF,
            'else': TokenType.ELSE,
            'while': TokenType.WHILE,
            'for': TokenType.FOR,
            'in': TokenType.IN,
            'range': TokenType.RANGE,
            'func': TokenType.FUNC,
            'return': TokenType.RETURN,
            'let': TokenType.LET,
            'continue': TokenType.CONTINUE,
            'break': TokenType.BREAK,
            'true': TokenType.TRUE,
            'false': TokenType.FALSE,
            'null': TokenType.NULL
        };
    }

    /**
     * Get the current character
     */
    currentChar() {
        if (this.position >= this.source.length) {
            return null;
        }
        return this.source[this.position];
    }
}
```

```
}

/** 
 * Look ahead at the next character without advancing the position
 */
peekChar() {
    if (this.position + 1 >= this.source.length) {
        return null;
    }
    return this.source[this.position + 1];
}

/** 
 * Advance to the next character
 */
advance() {
    if (this.currentChar() === '\n') {
        this.line++;
        this.column = 1;
    } else {
        this.column++;
    }
    this.position++;
}

/** 
 * Skip whitespace
 */
skipWhitespace() {
    while (this.currentChar() && /\s/.test(this.currentChar())) {
        this.advance();
    }
}

/** 
 * Skip a comment (everything from # to the end of the line)
 */
skipComment() {
    while (this.currentChar() && this.currentChar() !== '\n') {
        this.advance();
    }
    if (this.currentChar() === '\n') {
        this.advance();
    }
}

/** 
 * Tokenize a number
 */
tokenizeNumber() {
    let start = this.position;
    let startColumn = this.column;
    let hasDot = false;
```

```
while (this.currentChar() && /[0-9.]/.test(this.currentChar())) {
    if (this.currentChar() === '.') {
        if (hasDot) {
            break;
        }
        hasDot = true;
    }
    this.advance();
}

let value = this.source.substring(start, this.position);
return new Token(TokenType.NUMBER, parseFloat(value), this.line,
startColumn);
}

/**
 * Tokenize a string
 */
tokenizeString(quoteChar) {
    let start = this.position;
    let startColumn = this.column;
    let value = '';

    // Skip the opening quote
    this.advance();

    while (this.currentChar() && this.currentChar() !== quoteChar) {
        // Handle escape sequences
        if (this.currentChar() === '\\') {
            this.advance();
            if (this.currentChar() === 'n') {
                value += '\n';
            } else if (this.currentChar() === 't') {
                value += '\t';
            } else if (this.currentChar() === 'r') {
                value += '\r';
            } else if (this.currentChar() === quoteChar) {
                value += quoteChar;
            } else if (this.currentChar() === '\\') {
                value += '\\';
            } else {
                value += this.currentChar();
            }
        } else {
            value += this.currentChar();
        }
        this.advance();
    }

    // Skip the closing quote
    if (this.currentChar() === quoteChar) {
        this.advance();
    }
}
```

```
        return new Token(TokenType.STRING, value, this.line, startColumn);
    }

    /**
     * Tokenize an identifier or keyword
     */
    tokenizeIdentifier() {
        let start = this.position;
        let startColumn = this.column;

        while (this.currentChar() && /[A-Za-z0-9_]/.test(this.currentChar())) {
            this.advance();
        }

        let value = this.source.substring(start, this.position);

        // Check if it's a keyword
        let tokenType = this.keywords[value] || TokenType.IDENTIFIER;

        return new Token(tokenType, value, this.line, startColumn);
    }

    /**
     * Tokenize the input code
     */
    tokenize() {
        while (this.position < this.source.length) {
            // Skip whitespace
            if (/^\s/.test(this.currentChar())) {
                this.skipWhitespace();
                continue;
            }

            // Skip comments
            if (this.currentChar() === '#') {
                this.skipComment();
                continue;
            }

            // Numbers
            if (/^\d/.test(this.currentChar())) {
                this.tokens.push(this.tokenizeNumber());
                continue;
            }

            // Strings
            if (this.currentChar() === '"' || this.currentChar() === "'") {
                this.tokens.push(this.tokenizeString(this.currentChar()));
                continue;
            }

            // Identifiers and keywords
            if (/^\w/.test(this.currentChar())) {
                this.tokens.push(this.tokenizeIdentifier());
            }
        }
    }
}
```

```
        continue;
    }

    // Operators and punctuation
    switch (this.currentChar()) {
        case '+':
            this.tokens.push(new Token(TokenType.PLUS, '+', this.line,
this.column));
            this.advance();
            break;
        case '-':
            this.tokens.push(new Token(TokenType_MINUS, '-', this.line,
this.column));
            this.advance();
            break;
        case '*':
            this.tokens.push(new Token(TokenType.TIMES, '*', this.line,
this.column));
            this.advance();
            break;
        case '/':
            this.tokens.push(new Token(TokenType.DIVIDE, '/', this.line,
this.column));
            this.advance();
            break;
        case '%':
            this.tokens.push(new Token(TokenType.MODULO, '%', this.line,
this.column));
            this.advance();
            break;
        case '=':
            if (this.peekChar() === '=') {
                this.advance();
                this.advance();
                this.tokens.push(new Token(TokenType.EQUAL, '==',
this.line, this.column - 2));
            } else {
                this.advance();
                this.tokens.push(new Token(TokenType.ASSIGN, '=',
this.line, this.column - 1));
            }
            break;
        case '!':
            if (this.peekChar() === '=') {
                this.advance();
                this.advance();
                this.tokens.push(new Token(TokenType.NOT_EQUAL, '!=',
this.line, this.column - 2));
            } else {
                this.advance();
                this.tokens.push(new Token(TokenType.NOT, '!', this.line,
this.column - 1));
            }
            break;
    }
}
```

```
        case '<':
            if (this.peekChar() === '=') {
                this.advance();
                this.advance();
                this.tokens.push(new Token(TokenType.LESS_EQUAL, '<=',
this.line, this.column - 2));
            } else {
                this.advance();
                this.tokens.push(new Token(TokenType.LESS, '<', this.line,
this.column - 1));
            }
            break;
        case '>':
            if (this.peekChar() === '=') {
                this.advance();
                this.advance();
                this.tokens.push(new Token(TokenType.GREATER_EQUAL, '>=',
this.line, this.column - 2));
            } else {
                this.advance();
                this.tokens.push(new Token(TokenType.GREATER, '>', this.line,
this.column - 1));
            }
            break;
        case '&':
            if (this.peekChar() === '&') {
                this.advance();
                this.advance();
                this.tokens.push(new Token(TokenType.AND, '&&', this.line,
this.column - 2));
            } else {
                throw new Error(`Unexpected character '&' at line
${this.line}, column ${this.column}`);
            }
            break;
        case '|':
            if (this.peekChar() === '|') {
                this.advance();
                this.advance();
                this.tokens.push(new Token(TokenType.OR, '||', this.line,
this.column - 2));
            } else {
                throw new Error(`Unexpected character '|' at line
${this.line}, column ${this.column}`);
            }
            break;
        case '(':
            this.tokens.push(new Token(TokenType.LPAREN, '(', this.line,
this.column));
            this.advance();
            break;
        case ')':
            this.tokens.push(new Token(TokenType.RPAREN, ')', this.line,
this.column));
```

```
        this.advance();
        break;
    case '{':
        this.tokens.push(new Token(TokenType.LBRACE, '{', this.line,
this.column));
        this.advance();
        break;
    case '}':
        this.tokens.push(new Token(TokenType.RBRACE, '}', this.line,
this.column));
        this.advance();
        break;
    case '[':
        this.tokens.push(new Token(TokenType.LBRACKET, '[', this.line,
this.column));
        this.advance();
        break;
    case ']':
        this.tokens.push(new Token(TokenType.RBRACKET, ']', this.line,
this.column));
        this.advance();
        break;
    case ',':
        this.tokens.push(new Token(TokenType.COMMA, ',', this.line,
this.column));
        this.advance();
        break;
    case '.':
        this.tokens.push(new Token(TokenType.DOT, '.', this.line,
this.column));
        this.advance();
        break;
    case ':':
        this.tokens.push(new Token(TokenType.COLON, ':', this.line,
this.column));
        this.advance();
        break;
    case ';':
        this.tokens.push(new Token(TokenType.SEMICOLON, ';',
this.line, this.column));
        this.advance();
        break;
    default:
        throw new Error(`Unexpected character '${this.currentChar()}' at line ${this.line}, column ${this.column}`);
    }
}

// End of file
this.tokens.push(new Token(TokenType.EOF, null, this.line, this.column));

return this.tokens;
}
}
```

```
/*
 * Parser class for parsing tokens into an AST
 */
class Parser {
    constructor(tokens) {
        this.tokens = tokens;
        this.position = 0;
    }

    /**
     * Get the current token
     */
    currentToken() {
        if (this.position >= this.tokens.length) {
            return this.tokens[this.tokens.length - 1]; // Return EOF token
        }
        return this.tokens[this.position];
    }

    /**
     * Look ahead at the next token without advancing the position
     */
    peekToken() {
        if (this.position + 1 >= this.tokens.length) {
            return this.tokens[this.tokens.length - 1]; // Return EOF token
        }
        return this.tokens[this.position + 1];
    }

    /**
     * Advance to the next token
     */
    advance() {
        this.position++;
    }

    /**
     * Check if the current token is of the given type
     */
    check(type) {
        return this.currentToken().type === type;
    }

    /**
     * Consume a token of the given type and advance
     */
    consume(type, errorMessage) {
        if (this.check(type)) {
            const token = this.currentToken();
            this.advance();
            return token;
        }
    }
}
```

```
        throw new Error(`errorMessage at line ${this.currentToken().line},  
column ${this.currentToken().column}`);
    }

    /**
     * Parse the program
     */
    parse() {
        const statements = [];

        while (!this.check(TokenType.EOF)) {
            statements.push(this.parseStatement());
        }

        return statements;
    }

    /**
     * Parse a statement
     */
    parseStatement() {
        // Variable declaration with 'let'
        if (this.check(TokenType.LET)) {
            return this.parseVariableDeclaration();
        }

        // If statement
        if (this.check(TokenType.IF)) {
            return this.parseIfStatement();
        }

        // While loop
        if (this.check(TokenType.WHILE)) {
            return this.parseWhileLoop();
        }

        // For loop
        if (this.check(TokenType.FOR)) {
            return this.parseForLoop();
        }

        // Function declaration
        if (this.check(TokenType.FUNC)) {
            return this.parseFunctionDeclaration();
        }

        // Return statement
        if (this.check(TokenType.RETURN)) {
            return this.parseReturnStatement();
        }

        // Assignment (variable = expression)
        if (this.check(TokenType.IDENTIFIER) && this.peekToken().type ===  
TokenType.ASSIGN) {
```

```
        return this.parseAssignment();
    }

    // Expression statement
    return this.parseExpressionStatement();
}

/** 
 * Parse a variable declaration (let x = expr)
 */
parseVariableDeclaration() {
    this.consume(TokenType.LET, "Expected 'let'");
    const name = this.consume(TokenType.IDENTIFIER, "Expected variable name").value;
    this.consume(TokenType.ASSIGN, "Expected '=' after variable name");
    const expression = this.parseExpression();

    return {
        type: 'assign',
        variable: name,
        expression: expression
    };
}

/** 
 * Parse an assignment (x = expr)
 */
parseAssignment() {
    const name = this.consume(TokenType.IDENTIFIER, "Expected variable name").value;
    this.consume(TokenType.ASSIGN, "Expected '=' after variable name");
    const expression = this.parseExpression();

    return {
        type: 'assign',
        variable: name,
        expression: expression
    };
}

/** 
 * Parse an if statement (if expr { ... } else { ... })
 */
parseIfStatement() {
    this.consume(TokenType.IF, "Expected 'if'");
    const condition = this.parseExpression();

    this.consume(TokenType.LBRACE, "Expected '{' after if condition");
    const body = this.parseBlock();

    let elseBody = null;

    // Check for else block
    if (this.check(TokenType.ELSE)) {
```

```
        this.advance();
        this.consume(TokenType.LBRACE, "Expected '{' after 'else'");
        elseBody = this.parseBlock();
    }

    return {
        type: 'if',
        condition: condition,
        body: body,
        elseBody: elseBody
    };
}

/** 
 * Parse a while loop (while expr { ... })
 */
parseWhileLoop() {
    this.consume(TokenType.WHILE, "Expected 'while'");
    const condition = this.parseExpression();

    this.consume(TokenType.LBRACE, "Expected '{' after while condition");
    const body = this.parseBlock();

    return {
        type: 'while',
        condition: condition,
        body: body
    };
}

/** 
 * Parse a for loop (for i in range(start, end) { ... })
 */
parseForLoop() {
    this.consume(TokenType.FOR, "Expected 'for'");
    const variable = this.consume(TokenType.IDENTIFIER, "Expected variable name").value;

    this.consume(TokenType.IN, "Expected 'in'");
    this.consume(TokenType.RANGE, "Expected 'range'");
    this.consume(TokenType.LPAREN, "Expected '(' after 'range'");

    const start = this.parseExpression();
    this.consume(TokenType.COMMA, "Expected ',' between range values");
    const end = this.parseExpression();

    this.consume(TokenType.RPAREN, "Expected ')' after range values");
    this.consume(TokenType.LBRACE, "Expected '{' after for loop header");

    const body = this.parseBlock();

    return {
        type: 'for',
        variable: variable,
```

```
        start: start,
        end: end,
        body: body
    );
}

/** 
 * Parse a function declaration (func name(params) { ... })
 */
parseFunctionDeclaration() {
    this.consume(TokenType.FUNC, "Expected 'func'");
    const name = this.consume(TokenType.IDENTIFIER, "Expected function
name").value;

    this.consume(TokenType.LPAREN, "Expected '(' after function name");
    const params = this.parseParameters();
    this.consume(TokenType.RPAREN, "Expected ')' after parameters");

    this.consume(TokenType.LBRACE, "Expected '{' after function parameters");
    const body = this.parseBlock();

    return {
        type: 'function',
        name: name,
        params: params,
        body: body
    };
}

/** 
 * Parse function parameters
 */
parseParameters() {
    const params = [];

    // Empty parameter list
    if (this.check(TokenType.RPAREN)) {
        return params;
    }

    // Parse first parameter
    params.push(this.consume(TokenType.IDENTIFIER, "Expected parameter
name").value);

    // Parse remaining parameters
    while (this.check(TokenType.COMMA)) {
        this.advance();
        params.push(this.consume(TokenType.IDENTIFIER, "Expected parameter
name").value);
    }

    return params;
}
```

```
/*
 * Parse a block of statements
 */
parseBlock() {
    const statements = [];

    while (!this.check(TokenType.RBRACE) && !this.check(TokenType.EOF)) {
        statements.push(this.parseStatement());
    }

    this.consume(TokenType.RBRACE, "Expected '}'");

    return statements;
}

/*
 * Parse a return statement
 */
parseReturnStatement() {
    this.consume(TokenType.RETURN, "Expected 'return'");
    const expression = this.parseExpression();

    return {
        type: 'return',
        expression: expression
    };
}

/*
 * Parse an expression statement
 */
parseExpressionStatement() {
    const expression = this.parseExpression();

    return {
        type: 'expression',
        expression: expression
    };
}

/*
 * Parse an expression
 */
parseExpression() {
    return this.parseLogicalOr();
}

/*
 * Parse logical OR expressions
 */
parseLogicalOr() {
    let expr = this.parseLogicalAnd();

    while (this.check(TokenType.OR)) {
```

```
const operator = this.currentToken().value;
this.advance();
const right = this.parseLogicalAnd();

expr = {
    type: 'binary',
    operator: operator,
    left: expr,
    right: right
};

}

return expr;
}

/** 
 * Parse logical AND expressions
 */
parseLogicalAnd() {
    let expr = this.parseEquality();

    while (this.check(TokenType.AND)) {
        const operator = this.currentToken().value;
        this.advance();
        const right = this.parseEquality();

        expr = {
            type: 'binary',
            operator: operator,
            left: expr,
            right: right
        };
    }

    return expr;
}

/** 
 * Parse equality expressions (==, !=)
 */
parseEquality() {
    let expr = this.parseComparison();

    while (this.check(TokenType.EQUAL) || this.check(TokenType.NOT_EQUAL)) {
        const operator = this.currentToken().value;
        this.advance();
        const right = this.parseComparison();

        expr = {
            type: 'binary',
            operator: operator,
            left: expr,
            right: right
        };
    }
}
```

```
    }

    return expr;
}

/** 
 * Parse comparison expressions (<, >, <=, >=)
 */
parseComparison() {
    let expr = this.parseAddition();

    while (
        this.check(TokenType.LESS) ||
        this.check(TokenType.GREATER) ||
        this.check(TokenType.LESS_EQUAL) ||
        this.check(TokenType.GREATER_EQUAL)
    ) {
        const operator = this.currentToken().value;
        this.advance();
        const right = this.parseAddition();

        expr = {
            type: 'binary',
            operator: operator,
            left: expr,
            right: right
        };
    }

    return expr;
}

/** 
 * Parse addition and subtraction
 */
parseAddition() {
    let expr = this.parseMultiplication();

    while (this.check(TokenType.PLUS) || this.check(TokenType_MINUS)) {
        const operator = this.currentToken().value;
        this.advance();
        const right = this.parseMultiplication();

        expr = {
            type: 'binary',
            operator: operator,
            left: expr,
            right: right
        };
    }

    return expr;
}
```

```
/*
 * Parse multiplication, division, and modulo
 */
parseMultiplication() {
    let expr = this.parseUnary();

    while (
        this.check(TokenType.TIMES) ||
        this.check(TokenType.DIVIDE) ||
        this.check(TokenType.MODULO)
    ) {
        const operator = this.currentToken().value;
        this.advance();
        const right = this.parseUnary();

        expr = {
            type: 'binary',
            operator: operator,
            left: expr,
            right: right
        };
    }

    return expr;
}

/*
 * Parse unary expressions (!, -)
 */
parseUnary() {
    if (this.check(TokenType.NOT) || this.check(TokenType_MINUS)) {
        const operator = this.currentToken().value;
        this.advance();
        const right = this.parseUnary();

        return {
            type: 'unary',
            operator: operator,
            expression: right
        };
    }

    return this.parseCallOrPrimary();
}

/*
 * Parse function calls or primary expressions
 */
parseCallOrPrimary() {
    // Parse the primary expression first
    let expr = this.parsePrimary();

    // If followed by '(', it's a function call
    while (this.check(TokenType_LPAREN)) {
```

```
        expr = this.parseCall(expr);
    }

    return expr;
}

/**
 * Parse function call
 */
parseCall(callee) {
    // Function name must be an identifier
    if (callee.type !== 'variable') {
        throw new Error("Expected function name");
    }

    this.consume(TokenType.LPAREN, "Expected '(' after function name");
    const args = this.parseArguments();
    this.consume(TokenType.RPAREN, "Expected ')' after arguments");

    return {
        type: 'call',
        function: callee.name,
        arguments: args
    };
}

/**
 * Parse function call arguments
 */
parseArguments() {
    const args = [];

    // Empty argument list
    if (this.check(TokenType.RPAREN)) {
        return args;
    }

    // Parse first argument
    args.push(this.parseExpression());

    // Parse remaining arguments
    while (this.check(TokenType.COMMA)) {
        this.advance();
        args.push(this.parseExpression());
    }

    return args;
}

/**
 * Parse primary expressions (literals, identifiers, parenthesized
expressions)
*/
parsePrimary() {
```

```
// Number literal
if (this.check(TokenType.NUMBER)) {
    const value = this.currentToken().value;
    this.advance();
    return {
        type: 'value',
        value: value
    };
}

// String literal
if (this.check(TokenType.STRING)) {
    const value = this.currentToken().value;
    this.advance();
    return {
        type: 'value',
        value: value
    };
}

// Boolean literals
if (this.check(TokenType.TRUE)) {
    this.advance();
    return {
        type: 'value',
        value: true
    };
}

if (this.check(TokenType.FALSE)) {
    this.advance();
    return {
        type: 'value',
        value: false
    };
}

// Null literal
if (this.check(TokenType.NULL)) {
    this.advance();
    return {
        type: 'value',
        value: null
    };
}

// Identifier
if (this.check(TokenType.IDENTIFIER)) {
    const name = this.currentToken().value;
    this.advance();
    return {
        type: 'variable',
        name: name
    };
}
```

```
}

// Array literal
if (this.check(TokenType.LBRACKET)) {
    return this.parseArrayLiteral();
}

// Parenthesized expression
if (this.check(TokenType.LPAREN)) {
    this.advance();
    const expr = this.parseExpression();
    this.consume(TokenType.RPAREN, "Expected ')' after expression");
    return expr;
}

throw new Error(`Unexpected token ${this.currentToken().type} at line
${this.currentToken().line}, column ${this.currentToken().column}`);
}

/** 
 * Parse array literal
 */
parseArrayLiteral() {
    this.consume(TokenType.LBRACKET, "Expected '['");
    const elements = [];

    // Empty array
    if (this.check(TokenType.RBRACKET)) {
        this.advance();
        return {
            type: 'array',
            elements: elements
        };
    }

    // Parse first element
    elements.push(this.parseExpression());

    // Parse remaining elements
    while (this.check(TokenType.COMMA)) {
        this.advance();
        elements.push(this.parseExpression());
    }

    this.consume(TokenType.RBRACKET, "Expected ']'");
}

return {
    type: 'array',
    elements: elements
};
}

/**
```

```
* Main parser class for OperatorLang
*/
class OperatorLangParser {
    constructor(code, context) {
        this.code = code;
        this.context = context;
    }

    /**
     * Parse the code into a program
     */
    parse() {
        try {
            // Tokenize the input
            const lexer = new Lexer(this.code);
            const tokens = lexer.tokenize();

            // Parse the tokens into an AST
            const parser = new Parser(tokens);
            const program = parser.parse();

            // Process function declarations
            this.processFunctionDeclarations(program);

            // Return the program without function declarations
            return program.filter(statement => statement.type !== 'function');
        } catch (e) {
            this.context.writeToConsole(`Parse error: ${e.message}`);
            throw e;
        }
    }

    /**
     * Process function declarations
     */
    processFunctionDeclarations(program) {
        for (const statement of program) {
            if (statement.type === 'function') {
                this.context.defineFunction(statement.name, statement.params,
statement.body);
            }
        }
    }

    // Example programs
    const EXAMPLE_PROGRAMS = {
        helloWorld: `# Hello world program
print("Hello, world!")
`,

        drawingDemo: `# Graphics demo program
clearCanvas("#f0f0f0")
```

```
# Draw a colorful pattern
let size = 20
let colors = array("#ff0000", "#00ff00", "#0000ff", "#ffff00", "#ff00ff",
"#00ffff")

for i in range(0, 10) {
    for j in range(0, 10) {
        let x = i * 30 + 50
        let y = j * 30 + 50
        let colorIndex = (i + j) % 6
        setFillColor(get(colors, colorIndex))
        drawRect(x, y, size, size)
    }
}

# Draw a rainbow arc
setLineWidth(5)
let cx = 200
let cy = 200
let radius = 100
let colors2 = array("#ff0000", "#ff7f00", "#ffff00", "#00ff00", "#0000ff",
"#4b0082", "#9400d3")

for i in range(0, 7) {
    let r = radius - i * 10
    setStrokeColor(get(colors2, i))
    drawCircle(cx, cy, r, false)
}

# Draw some text
setFillColor("#000000")
setFont("20px Arial")
drawText("Operator Graphics Demo", 120, 350)
`,

fibonacci: `# Calculate Fibonacci sequence
# Iterative implementation to avoid recursion issues

print("Fibonacci Sequence:")

# Initialize first two Fibonacci numbers
let a = 0
let b = 1

# Calculate and display the first 10 Fibonacci numbers
for i in range(0, 10) {
    # Use separate if statements instead of else if
    if i == 0 {
        print("fibonacci(0) = 0")
    }

    if i == 1 {
        print("fibonacci(1) = 1")
    }
}
```

```
if i > 1 {
    # Calculate the next Fibonacci number
    let c = a + b

    # Update a and b for the next iteration
    a = b
    b = c

    # Print the result
    print("fibonacci(" + i + ") = " + c)
}

print("Fibonacci sequence completed")
`,

gameOfLife: `# Conway's Game of Life - Simplified Version
# This implementation uses a flat 1D array instead of nested arrays

# Grid dimensions
let width = 20
let height = 15
let cellSize = 15

# Debug output
func debug(message) {
    print("DEBUG: " + message)
}

# Convert 2D coordinates to 1D index
func coordToIndex(x, y) {
    return y * width + x
}

# Initialize a random grid
func createGrid() {
    debug("Creating grid...")

    # Create a flat array for the grid
    let grid = array()

    # Fill with random cells
    for i in range(0, width * height) {
        let state = 0
        if random(0, 1) < 0.3 {
            state = 1
        }
        grid = push(grid, state)
    }

    debug("Grid created with " + length(grid) + " cells")
    return grid
}
```

```
# Get cell state
func getCell(grid, x, y) {
    # Check bounds
    if x < 0 {
        return 0
    }

    if x >= width {
        return 0
    }

    if y < 0 {
        return 0
    }

    if y >= height {
        return 0
    }

    # Get index
    let index = coordToIndex(x, y)

    # Check if grid and index are valid
    if grid == null {
        return 0
    }

    # Get the cell value
    let cell = get(grid, index)

    # Return 0 if cell is not defined
    if cell == null {
        return 0
    }

    return cell
}

# Count neighbors
func countNeighbors(grid, x, y) {
    let count = 0

    # Check all 8 surrounding cells
    for dy in range(-1, 2) {
        for dx in range(-1, 2) {
            # Skip the cell itself (when dx=0 and dy=0)
            # We'll handle this with an if statement instead of continue
            if dx != 0 || dy != 0 {
                # Add to count if neighbor is alive
                let nx = x + dx
                let ny = y + dy

                if getCell(grid, nx, ny) == 1 {
```

```
        count = count + 1
    }
}
}

return count
}

# Compute next generation
func nextGeneration(grid) {
    debug("Computing next generation...")

    # Create new grid
    let newGrid = array()

    # Initialize with all dead cells
    for i in range(0, width * height) {
        newGrid = push(newGrid, 0)
    }

    # Apply rules to each cell
    for y in range(0, height) {
        for x in range(0, width) {
            # Get current state
            let state = getCell(grid, x, y)

            # Count neighbors
            let neighbors = countNeighbors(grid, x, y)

            # Index in the flat array
            let index = coordToIndex(x, y)

            # Apply Conway's rules
            let newState = 0

            # Rule 1: Live cell with 2-3 neighbors survives
            if state == 1 {
                if neighbors == 2 {
                    newState = 1
                }

                if neighbors == 3 {
                    newState = 1
                }
            }

            # Rule 2: Dead cell with 3 neighbors becomes alive
            if state == 0 {
                if neighbors == 3 {
                    newState = 1
                }
            }
        }
    }
}
```

```
        # Update the new grid
        newGrid = set(newGrid, index, newState)
    }
}

debug("Next generation computed")
return newGrid
}

# Draw the grid
func drawGrid(grid) {
    debug("Drawing grid...")

    # Clear the canvas
    clearCanvas("#f0f0f0")

    # Draw each cell
    for y in range(0, height) {
        for x in range(0, width) {
            # Get cell state
            let state = getCell(grid, x, y)

            # Set color based on state
            if state == 1 {
                setFillColor("#000000") # Black for alive
            } else {
                setFillColor("#ffffff") # White for dead
            }

            # Draw the cell
            drawRect(x * cellSize + 1, y * cellSize + 1,
                     cellSize - 2, cellSize - 2)
        }
    }

    # Draw grid lines
    setStrokeColor("#cccccc")
    setLineWidth(1)

    # Vertical lines
    for x in range(0, width + 1) {
        drawLine(x * cellSize, 0, x * cellSize, height * cellSize)
    }

    # Horizontal lines
    for y in range(0, height + 1) {
        drawLine(0, y * cellSize, width * cellSize, y * cellSize)
    }

    debug("Grid drawn")
}

# Start simulation
print("Conway's Game of Life")
```

```
print("Initializing...")

# Create and draw initial grid
let grid = createGrid()
drawGrid(grid)

print("Simulation running...")
print("Press 'Run' again to stop")

# Animation loop
while true {
    # Update grid
    grid = nextGeneration(grid)

    # Draw updated grid
    drawGrid(grid)

    # Delay for animation
    delay(200)
}

```
};

// Set up the OperatorLanguage UI with the Operator System
document.addEventListener('DOMContentLoaded', function() {
    // Add the programming language section to the main UI
    const mainElement = document.querySelector('main');

    if (!mainElement) {
        console.error('Main element not found in the document');
        return;
    }

    // Create programming language section
    const programmingSection = document.createElement('section');
    programmingSection.id = 'programming-section';
    programmingSection.className = 'operation-section';
    programmingSection.innerHTML = `
        <h2>Programmable Interface</h2>
        <div class="operation-container">
            <div class="operation-controls">
                <h3>OperatorLang Code</h3>
                <textarea id="code-editor" spellcheck="false"></textarea>
                <div class="editor-toolbar">
                    <button id="run-program" class="button">Run</button>
                    <button id="stop-program" class="button button-secondary">Stop</button>
                    <button id="step-program" class="button">Step</button>
                    <button id="clear-program" class="button button-secondary">Clear</button>
                </div>
                <select id="example-programs">
                    <option value="">Load Example...</option>
                    <option value="helloWorld">Hello World</option>
                    <option value="drawingDemo">Drawing Demo</option>
                </select>
            </div>
        </div>
    `;

    mainElement.appendChild(programmingSection);
});
```

```
        <option value="fibonacci">Fibonacci Sequence</option>
        <option value="gameOfLife">Game of Life</option>
    </select>
</div>
<div class="language-console-container">
    <h3>Console Output</h3>
    <div id="language-console" class="language-console"></div>
</div>
<div class="operation-result">
    <h3>Graphics Output</h3>
    <canvas id="language-canvas" width="400" height="400"></canvas>
    <div class="program-controls">
        <button id="save-program" class="button">Save Program</button>
        <button id="load-program" class="button">Load Program</button>
        <button id="download-canvas" class="button">Download
Image</button>
    </div>
</div>
</div>
`;

// Add programming section to the main UI
mainElement.appendChild(programmingSection);

// Add it to navigation
const navSection = document.querySelector('#main-nav');
if (navSection) {
    const programmingNav = document.createElement('div');
    programmingNav.className = 'nav-section';
    programmingNav.innerHTML =
        `<h3>Programming</h3>
        <button data-section="programming-section">OperatorLang</button>
`;
    navSection.appendChild(programmingNav);
}

// Add the styles
const styleElement = document.createElement('style');
styleElement.textContent =
`
#code-editor {
    width: 100%;
    height: 300px;
    font-family: 'Fira Code', 'Consolas', monospace;
    font-size: 14px;
    padding: 10px;
    border: 1px solid var(--secondary-color);
    resize: vertical;
    tab-size: 4;
    white-space: pre;
    overflow-wrap: normal;
    overflow-x: auto;
}
`;
```

```
.editor-toolbar {
    display: flex;
    gap: 10px;
    margin-top: 10px;
    flex-wrap: wrap;
}

.language-console {
    background-color: var(--terminal-bg);
    color: var(--terminal-text);
    height: 150px;
    overflow-y: auto;
    padding: 10px;
    font-family: 'Consolas', 'Courier New', monospace;
    margin-top: 10px;
    border: 1px solid var(--secondary-color);
}

.language-console-container {
    margin-top: 15px;
}

#language-canvas {
    border: 1px solid var(--secondary-color);
    background-color: white;
    width: 100%;
    height: 400px;
    max-width: 400px;
}

.program-controls {
    display: flex;
    gap: 10px;
    margin-top: 15px;
    flex-wrap: wrap;
}

.console-line {
    margin-bottom: 5px;
    word-wrap: break-word;
}
};

document.head.appendChild(styleElement);

// Initialize the language
const lang = new OperatorLanguage('language-canvas', 'language-console');
window.operatorLang = lang; // Make it globally accessible

// Set up event listeners
const codeEditor = document.getElementById('code-editor');
const runButton = document.getElementById('run-program');
const stopButton = document.getElementById('stop-program');
const stepButton = document.getElementById('step-program');
const clearButton = document.getElementById('clear-program');
```

```
const exampleSelect = document.getElementById('example-programs');
const saveButton = document.getElementById('save-program');
const loadButton = document.getElementById('load-program');
const downloadButton = document.getElementById('download-canvas');

if (!codeEditor || !runButton || !stopButton || !stepButton || !clearButton ||
!exampleSelect || !saveButton || !loadButton || !downloadButton) {
    console.error('One or more UI elements not found');
    return;
}

// Run program
runButton.addEventListener('click', function() {
    if (lang.running) {
        lang.stop();
        runButton.textContent = 'Run';
    } else {
        const code = codeEditor.value;
        try {
            lang.loadProgram(code);
            lang.run();
            runButton.textContent = 'Pause';
        } catch (error) {
            lang.writeToConsole(`Error loading program: ${error.message}`);
        }
    }
});

// Stop program
stopButton.addEventListener('click', function() {
    lang.stop();
    runButton.textContent = 'Run';
});

// Step program
stepButton.addEventListener('click', function() {
    const code = codeEditor.value;
    if (lang.program.length === 0) {
        try {
            lang.loadProgram(code);
        } catch (error) {
            lang.writeToConsole(`Error loading program: ${error.message}`);
            return;
        }
    }
    lang.step();
});

// Clear program
clearButton.addEventListener('click', function() {
    if (confirm('Clear the code editor?')) {
        codeEditor.value = '';
        lang.clearConsole();
    }
})
```

```
});

// Load example
exampleSelect.addEventListener('change', function() {
    const example = exampleSelect.value;
    if (example && EXAMPLE_PROGRAMS[example]) {
        if (codeEditor.value.trim() !== '' &&
            !confirm('Replace current code with example?')) {
            exampleSelect.value = '';
            return;
        }
        codeEditor.value = EXAMPLE_PROGRAMS[example];
        lang.clearConsole();
        exampleSelect.value = '';
    }
});

// Save program
saveButton.addEventListener('click', function() {
    try {
        // Create the program data
        const programData = {
            code: codeEditor.value,
            timestamp: new Date().toISOString(),
            version: '1.0'
        };

        // Save to localStorage
        localStorage.setItem('operatorLangProgram',
JSON.stringify(programData));

        // Create download file
        const blob = new Blob([JSON.stringify(programData, null, 2)], {type:
'application/json'});
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = 'operator_program.json';
        a.click();
        URL.revokeObjectURL(url);

        lang.writeToConsole('Program saved successfully');
    } catch (e) {
        lang.writeToConsole(`Error saving program: ${e.message}`);
    }
});

// Load program
loadButton.addEventListener('click', function() {
    // Create file input for loading JSON
    const fileInput = document.createElement('input');
    fileInput.type = 'file';
    fileInput.accept = '.json';
    fileInput.style.display = 'none';
});
```

```
document.body.appendChild(fileInput);

fileInput.addEventListener('change', function(e) {
    const file = e.target.files[0];
    if (!file) return;

    const reader = new FileReader();
    reader.onload = function(e) {
        try {
            const data = JSON.parse(e.target.result);

            // Validate the data
            if (!data.code || !data.version) {
                throw new Error('Invalid program file format');
            }

            if (data.version !== '1.0') {
                throw new Error(`Unsupported program version:
${data.version}`);
            }
        }

        // Set the code
        codeEditor.value = data.code;

        // Save to localStorage
        localStorage.setItem('operatorLangProgram',
JSON.stringify(data));

        lang.writeToConsole(`Program loaded from file (${new
Date(data.timestamp).toLocaleString()})`);
    } catch (e) {
        lang.writeToConsole(`Error loading program: ${e.message}`);
    }
};

reader.onerror = function() {
    lang.writeToConsole('Error reading file');
};
reader.readAsText(file);

document.body.removeChild(fileInput);
});

// Check if we have a saved program in localStorage
const savedProgram = localStorage.getItem('operatorLangProgram');
if (savedProgram) {
    try {
        const data = JSON.parse(savedProgram);

        // Validate the data
        if (!data.code || !data.version) {
            throw new Error('Invalid saved program format');
        }

        if (data.version !== '1.0') {
```

```
        throw new Error(`Unsupported program version:
${data.version}`);
    }

    // Prompt to load the saved program
    if (confirm(`Load saved program from ${new
Date(data.timestamp).toLocaleString()}?`)) {
        codeEditor.value = data.code;
        lang.writeToConsole(`Program loaded from local storage (${new
Date(data.timestamp).toLocaleString()})`);
        return;
    }
} catch (e) {
    lang.writeToConsole(`Error loading saved program: ${e.message}`);
}
});

// If no saved program or user declined, show file dialog
fileInput.click();
});

// Download canvas as image
downloadButton.addEventListener('click', function() {
    const canvas = document.getElementById('language-canvas');
    const url = canvas.toDataURL('image/png');
    const a = document.createElement('a');
    a.href = url;
    a.download = 'operator_canvas.png';
    a.click();
});

// Load any saved program on startup
const savedProgram = localStorage.getItem('operatorLangProgram');
if (savedProgram) {
    try {
        const data = JSON.parse(savedProgram);
        if (data.code && data.version === '1.0') {
            codeEditor.value = data.code;
        }
    } catch (e) {
        console.error('Error loading saved program:', e);
    }
} else {
    // Load a default program if no saved program exists
    codeEditor.value = EXAMPLE_PROGRAMS.helloWorld;
}

// Handle key shortcuts in the editor
codeEditor.addEventListener('keydown', function(e) {
    if (e.key === 'Tab') {
        e.preventDefault();

        // Insert a tab at the cursor position
        const start = this.selectionStart;
```

```
const end = this.selectionEnd;

// If multiple lines are selected, indent all of them
if (start !== end) {
    const lines = this.value.split('\n');
    let startLine = this.value.substr(0, start).split('\n').length - 1;
    let endLine = this.value.substr(0, end).split('\n').length - 1;

    let newStart = start;
    let newEnd = end;

    if (e.shiftKey) {
        // Unindent
        for (let i = startLine; i <= endLine; i++) {
            if (lines[i].startsWith('\t')) {
                lines[i] = lines[i].substring(1);
                if (i === startLine) {
                    newStart--;
                }
                newEnd--;
            }
        }
    } else {
        // Indent
        for (let i = startLine; i <= endLine; i++) {
            lines[i] = '\t' + lines[i];
            if (i === startLine) {
                newStart++;
            }
            newEnd++;
        }
    }
}

this.value = lines.join('\n');
this.selectionStart = newStart;
this.selectionEnd = newEnd;
} else {
    // Just insert a tab at cursor position
    this.value = this.value.substring(0, start) + '\t' +
    this.value.substring(end);
    this.selectionStart = this.selectionEnd = start + 1;
}
}
});
```

## [97] integerDatabase/operator-map.js

- **Bytes:** 36234
- **Type:** text

```
/**  
 * operator-map.js  
 *  
 * This module implements the Map System for the Operator framework,  
 * providing charset management and string ID calculation functionality.  
 * It integrates with the operator-dropdown.js system.  
 *  
 * @module OperatorMap  
 * @version 1.0.0  
 * @depends OperatorFramework.Dropdown  
 */  
  
// Access the shared namespace for Operator framework  
  
/**  
 * Map System for charset operations and string ID calculations  
 * @namespace OperatorMap  
 */  
OperatorFramework.Map = (function() {  
    'use strict';  
  
    // Private module variables  
    let _initialized = false;  
    let _section = null;  
    let _charsetConfig = [];  
    let _uniqueCharset = [];  
    let _defaultOptions = {  
        containerId: 'main', // ID of the container to add the map section to  
        sectionId: 'map-system-section' // ID for the created section element  
    };  
    let _options = {};  
  
    /**  
     * Initialize the Map System  
     * @param {Object} options - Configuration options  
     * @param {string} [options.containerId='main'] - ID of the container to add  
     * the map section to  
     * @param {string} [options.sectionId='map-system-section'] - ID for the  
     * created section element  
     * @returns {boolean} Success status  
     */  
    function initialize(options = {}) {  
        if (_initialized) {  
            console.warn('OperatorMap is already initialized');  
            return false;  
        }  
  
        // Merge options with defaults  
        _options = { ..._defaultOptions, ...options };  
  
        // Initialize the character set configuration  
        _initCharsetConfig();  
    }  
});
```

```
// Generate the unique charset based on the configuration
_uniqueCharset = generateCharsetFromConfig(_charsetConfig);

    console.log(`OperatorMap initialized with ${_uniqueCharset.length} characters`);
    _initialized = true;

    // Register with dropdown system if available
    if (OperatorFramework.Dropdown) {
        OperatorFramework.Dropdown.addSystem({
            id: 'map',
            name: 'Charset App',
            description: 'Calculate String IDs and perform charset operations',
            loadFunction: loadMapSystem
        });
        console.log('Map System registered with OperatorFramework.Dropdown');
    } else {
        console.warn('OperatorFramework.Dropdown not available, Map System not registered');
    }

    return true;
}

/**
 * Load the Map System UI and functionality
 * @returns {boolean} Success status
 */
function loadMapSystem() {
    if (!_initialized) {
        console.error('OperatorMap not initialized, call initialize() first');
        return false;
    }

    // Check if already loaded
    if (_section && document.getElementById(_options.sectionId)) {
        showMapSection();
        return true;
    }

    // Get container
    //const container = document.getElementById(_options.containerId);
    // Use:
    const container = document.querySelector('main');
    if (!container) {
        console.error(`Container with ID "${_options.containerId}" not found`);
        return false;
    }

    // Create Map System section
    _section = document.createElement('section');
    _section.id = _options.sectionId;
```

```
_section.className = 'operation-section';

// Populate with Map System content
_section.innerHTML = _getMapSystemHTML();

// Add to container
container.appendChild(_section);

// Add styles
_addMapSystemStyles();

// Initialize event listeners
_initMapSystemEventListeners();

// Show the section
showMapSection();

console.log('Map System loaded successfully');
return true;
}

/**
 * Show the Map System section and hide other sections
 */
function showMapSection() {
    if (!_section) return;

    // Hide all sections
    const allSections = document.querySelectorAll('.operation-section');
    allSections.forEach(section => {
        section.classList.remove('active');
    });

    // Show Map section
    _section.classList.add('active');
}

/**
 * Generate the Map System HTML content
 * @private
 * @returns {string} HTML content
 */
function _getMapSystemHTML() {
    return `
        <h2>Charset App (Spatial Systems Engineering)</h2>
        <div class="operation-container">
            <div class="operation-controls">
                <div id="main-menu">
                    <h3>Choose an option:</h3>
                    <button id="show-generate-combinations">Generate
Combinations</button>
                    <button id="show-calculate-string-id">Calculate String
ID</button>
                    <button id="show-decode-id">Decode ID to String</button>
    
```

```
        <button id="show-find-optimal-variable">Find Optimal  
Variable and Save to File</button>  
    </div>  
  
    <div id="color-bar" style="width: 100%; height: 20px;  
background-color: gold; margin-top: 10px;"></div>  
  
    <div id="option-container" style="display: none;">  
        <!-- Mode 1: Generate Combinations -->  
        <div id="generate-combinations" style="display: none;">  
            <h3>Generate Combinations</h3>  
            <label for="combination-size">Enter the size of  
combinations (n):</label>  
            <input type="number" id="combination-size">  
            <br>  
            <label for="output-file-generate">Enter the name of  
the output file:</label>  
            <input type="text" id="output-file-generate">  
            <br>  
            <label for="use-multithreading">Use Multithreading:  
        </label>  
        <select id="use-multithreading">  
            <option value="yes">Yes</option>  
            <option value="no">No</option>  
        </select>  
        <br>  
        <button id="generate-combinations-  
btn">Generate</button>  
    </div>  
  
        <!-- Mode 2: Calculate String ID -->  
        <div id="calculate-string-id" style="display: none;">  
            <h3>Calculate String ID</h3>  
            <label for="custom-string">Enter your custom string:  
        </label>  
        <textarea id="custom-string" rows="10" cols="50"  
wrap="off"></textarea>  
  
        <br>  
        <button id="calculate-string-id-  
btn">Calculate</button>  
        <p id="string-id-result"></p>  
        <div id="character-word-count">Characters: 0, Words:  
0</div>  
    </div>  
  
        <!-- Mode 3: Decode ID to String -->  
        <div id="decode-id" style="display: none;">  
            <h3>Decode ID to String</h3>  
            <label for="string-id">Enter the ID to decode:</label>  
            <input type="number" id="string-id">  
            <br>  
            <button id="decode-id-btn">Decode</button>  
            <p id="decoded-string-result"></p>  
        </div>
```

```
        <!-- Mode 4: Find Optimal Variable -->
        <div id="find-optimal-variable" style="display: none;">
            <h3>Find Optimal Variable and Save to File</h3>
            <label for="user-number">Enter the user number:<br>
                <input type="number" id="user-number">
            <br>
            <label for="output-file-optimal">Enter the name of the output file:</label>
                <input type="text" id="output-file-optimal">
            <br>
            <button id="find-optimal-variable-btn">Find and Save</button>
        </div>
    </div>
</div>

        <div class="operation-result">
            <h3>Map System Result</h3>
            <div id="map-result-display" class="result-container">
                <p>Select an operation from the left panel to start.</p>
            </div>
        </div>
    </div>
`;
}

/** 
 * Add CSS styles for the Map System
 * @private
 */
function _addMapSystemStyles() {
    const styleId = 'operator-map-styles';

    // Don't add styles if they already exist
    if (document.getElementById(styleId)) return;

    const styleElement = document.createElement('style');
    styleElement.id = styleId;
    styleElement.textContent =
        /* Basic Reset */
        '#map-system-section * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        /* Body Styling - adapted for the section */
        #map-system-section {
            font-family: "Fira Code", "Consolas", monospace;
            color: #333333; /* Dark Gray */
        }
}
```

```
/* Heading Styles */
#map-system-section h3 {
    color: #4B5320; /* Army Green */
    margin-bottom: 15px;
    text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.3);
    font-weight: 600;
}

/* Button Styles */
#map-system-section button {
    background-color: #4B5320; /* Army Green */
    color: #FFFFFF;
    padding: 10px 20px;
    font-size: 16px;
    border: 2px solid #BDB76B; /* Dark Khaki */
    cursor: pointer;
    transition: background-color 0.3s ease, color 0.3s ease;
    font-family: "Fira Code", "Consolas", monospace;
    margin-top: 10px;
}

#map-system-section button:hover {
    background-color: #BDB76B; /* Dark Khaki */
    color: #333333; /* Dark Gray */
}

/* Input and Textarea Styles */
#map-system-section input[type="text"],
#map-system-section input[type="number"],
#map-system-section textarea,
#map-system-section select {
    width: calc(100% - 20px);
    padding: 10px;
    font-size: 16px;
    background-color: #F5F5DC; /* Beige */
    color: #333333; /* Dark Gray */
    border: 2px solid #BDB76B; /* Dark Khaki */
    font-family: "Fira Code", "Consolas", monospace;
    margin-bottom: 10px;
}

#map-system-section textarea {
    resize: none; /* Disables resizing of the textarea */
}

/* Results Styling */
#string-id-result, #decoded-string-result {
    background-color: #FFFFFF; /* White */
    padding: 10px;
    border: 2px solid #BDB76B; /* Dark Khaki */
    font-family: "Fira Code", "Consolas", monospace;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    color: #333333; /* Dark Gray */
    word-wrap: break-word;
}
```

```
white-space: pre-wrap; /* Preserves spaces and line breaks */
margin-top: 10px;
}

/* Color bar styling */
#color-bar {
    width: 100%;           /* Full width */
    height: 20px;          /* Fixed height */
    background-color: #BDB76B; /* Dark Khaki */
    margin-top: 10px;        /* Space above the bar */
    border: 2px solid #4B5320; /* Army Green border */
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3); /* Subtle shadow */
    transition: background-color 0.3s ease; /* Smooth transition for
color change */
}
`;

document.head.appendChild(styleElement);
}

/**
 * Initialize the charset configuration
 * @private
 */
function _initCharsetConfig() {
    _charsetConfig = [
        { name: 'Basic Latin', range: [0x0020, 0x007F] },
        { name: 'Latin-1 Supplement', range: [0x0080, 0x00FF] },
        { name: 'Latin Extended-A', range: [0x0100, 0x017F] },
        { name: 'Latin Extended-B', range: [0x0180, 0x024F] },
        { name: 'Greek and Coptic', range: [0x0370, 0x03FF] },
        { name: 'Cyrillic', range: [0x0400, 0x04FF] },
        { name: 'Arabic', range: [0x0600, 0x06FF] },
        { name: 'Hebrew', range: [0x0590, 0x05FF] },
        { name: 'Devanagari', range: [0x0900, 0x097F] },
        { name: 'Mathematical Operators', range: [0x2200, 0x22FF] },
        { name: 'Supplemental Mathematical Operators', range: [0x2A00, 0x2AFF] },
    ],
    { name: 'Miscellaneous Technical', range: [0x2300, 0x23FF] },
    { name: 'Miscellaneous Symbols and Arrows', range: [0x2190, 0x21FF] },
    { name: 'CJK Unified Ideographs', range: [0x4E00, 0x9FFF] },
    { name: 'Hangul Syllables', range: [0xAC00, 0xD7AF] },
    { name: 'Hiragana', range: [0x3040, 0x309F] },
    { name: 'Katakana', range: [0x30A0, 0x30FF] },
    { name: 'Bopomofo', range: [0x3100, 0x312F] },
    { name: 'Currency Symbols', range: [0x20A0, 0x20CF] },
    { name: 'Additional Punctuation', range: [0x2000, 0x206F] }
];
}

/**
 * Generate a charset from configuration
 * @param {Array} config - Array of charset range configurations
 * @returns {Array} Array of characters in the charset

```

```
/*
function generateCharsetFromConfig(config) {
    const charset = new Set();
    config.forEach(block => {
        for (let i = block.range[0]; i <= block.range[1]; i++) {
            try {
                charset.add(String.fromCharCode(i));
            } catch (e) {
                console.error(`Failed to add character code ${i}:
${e.message}`);
            }
        }
    });
    // Explicitly add newline and tab characters
    charset.add('\n');
    charset.add('\t');
    return Array.from(charset);
}

/**
 * Initialize event listeners for the Map System
 * @private
 */
function _initMapSystemEventListeners() {
    // Option buttons
    document.getElementById('show-generate-
combinations').addEventListener('click', () => showOption('generate-
combinations'));
    document.getElementById('show-calculate-string-
id').addEventListener('click', () => showOption('calculate-string-id'));
    document.getElementById('show-decode-id').addEventListener('click', () =>
showOption('decode-id'));
    document.getElementById('show-find-optimal-
variable').addEventListener('click', () => showOption('find-optimal-variable'));

    // Action buttons
    document.getElementById('generate-combinations-
btn').addEventListener('click', generateCombinations);
    document.getElementById('calculate-string-id-
btn').addEventListener('click', calculateStringID);
    document.getElementById('decode-id-btn').addEventListener('click',
decodeID);
    document.getElementById('find-optimal-variable-
btn').addEventListener('click', findOptimalVariable);

    // Text area for character count
    const customStringTextarea = document.getElementById('custom-string');
    if (customStringTextarea) {
        customStringTextarea.addEventListener('input', function(event) {
            const inputString = event.target.value;
            displayCharacterAndWordCount(inputString);
        });
    }
}
```

```
/**  
 * Show an option panel and hide others  
 * @param {string} optionId - ID of the option to show  
 */  
function showOption(optionId) {  
    const options = document.querySelectorAll('#option-container > div');  
    options.forEach(opt => opt.style.display = 'none');  
    document.getElementById(optionId).style.display = 'block';  
    document.getElementById('option-container').style.display = 'block';  
}  
  
/**  
 * Generate combinations  
 */  
function generateCombinations() {  
    try {  
        const n = BigInt(document.getElementById('combination-size').value);  
        const outputFile = document.getElementById('output-file-  
generate').value || 'combinations.txt';  
  
        if (isNaN(Number(n)) || n <= 0n) {  
            throw new Error("Invalid combination size. Please enter a positive  
integer.");  
        }  
  
        // For large n values, show a warning  
        if (n > 4n) {  
            if (!confirm(`This will generate up to ${_uniqueCharset.length **  
Number(n)} combinations. This could be a large file. Continue?`)) {  
                return;  
            }  
        }  
  
        const resultDisplay = document.getElementById('map-result-display');  
        resultDisplay.innerHTML = '<p>Generating combinations... This may take  
a while for large values.</p>';  
  
        // Use setTimeout to allow UI to update before starting computation  
        setTimeout(() => {  
            try {  
                let combinationCount = 0;  
                let combinations = '';  
                const k = BigInt(_uniqueCharset.length);  
                const nbrComb = k ** n;  
  
                // For large combinations, limit how many we generate  
                const maxToGenerate = 1000000n; // 1 million max  
                const actualGenerate = nbrComb > maxToGenerate ? maxToGenerate  
: nbrComb;  
  
                // Generate combinations  
                for (let i = 0n; i < actualGenerate; i++) {  
                    let id = i;  
                }  
            } catch (error) {  
                console.error(error);  
            }  
        }, 0);  
    } catch (error) {  
        console.error(error);  
    }  
}
```

```
        let combination = '';
        for (let j = 0n; j < n; j++) {
            combination = _uniqueCharset[Number(id % k)] +
combination;
            id = id / k;
        }
        combinations += combination + '\n';
        combinationCount++;
    }

    // Download the file
    downloadFile(outputFile, combinations);

    // Update the result display
    resultDisplay.innerHTML =
        `<p>Successfully generated
${combinationCount.toLocaleString()} combinations.</p>
<p>Total possible combinations: ${nbrComb.toString()}.</p>
<p>File saved as: ${outputFile}</p>
`;

} catch (error) {
    resultDisplay.innerHTML = `<p class="error">Error during
generation: ${error.message}</p>`;
}
}, 100);
} catch (error) {
    alert(`Error: ${error.message}`);
}
}

/** 
 * Calculate a string ID
 */
function calculateStringID() {
    try {
        const inputString = document.getElementById('custom-string').value;
        if (inputString.trim() === '') {
            throw new Error("Input string cannot be empty.");
        }

        let id = 0n;
        const resultDisplay = document.getElementById('map-result-display');
        resultDisplay.innerHTML = '<p>Calculating string ID...</p>';

        // Use setTimeout to allow UI to update before starting computation
        setTimeout(() => {
            try {
                // Check all characters are in charset
                const invalidChars = [];
                for (const char of inputString) {
                    const charIndex = _uniqueCharset.indexOf(char);
                    if (charIndex === -1) {
                        invalidChars.push(char);
                    }
                }
            }
        }, 100);
    }
}
```

```
        }

        if (invalidChars.length > 0) {
            throw new Error(`Invalid characters found:
${invalidChars.join(' ')}`);
        }

        // Calculate ID
        for (const char of inputString) {
            const charIndex = _uniqueCharset.indexOf(char);
            id = id * BigInt(_uniqueCharset.length) +
BigInt(charIndex);
        }

        const result = id.toString() + "\t\n\n" + inputString;
        document.getElementById('string-id-result').innerText = "The
ID for the string is: " + id.toString();

        // Display character and word count
        displayCharacterAndWordCount(inputString);

        // Decode ID to color indexes and update color bar
        const colorIndexes = decodeIDtoColorIndexes(id.toString());
        updateColorBar(colorIndexes);

        // Update result display
        resultDisplay.innerHTML =
            <p>Calculation completed successfully.</p>
            <p>String ID: ${id.toString()}</p>
            <p>Character count: ${inputString.length}</p>
            <p>Color indexes: ${colorIndexes.length}</p>
            <p><button id="download-id-result">Download
Result</button></p>
        `;

        // Add event listener for download button
        document.getElementById('download-id-
result').addEventListener('click', () => {
            const outputFile = `${colorIndexes.length}.txt`;
            downloadFile(outputFile, result);
        });
    } catch (error) {
        resultDisplay.innerHTML = `<p class="error">Error calculating
ID: ${error.message}</p>`;
    }
}, 100);
} catch (error) {
    alert(`Error: ${error.message}`);
}
}

/**
 * Decode an ID to a string
 */
```

```
function decodeID() {
    try {
        let id = BigInt(document.getElementById('string-id').value);
        if (id < 0n) {
            throw new Error("ID cannot be negative.");
        }

        const resultDisplay = document.getElementById('map-result-display');
        resultDisplay.innerHTML = '<p>Decoding ID to string...</p>';

        // Use setTimeout to allow UI to update before starting computation
        setTimeout(() => {
            try {
                const decodedString = [];
                const originalId = id;

                // If ID is too large, show warning
                if (id > 10n ** 30n) {
                    resultDisplay.innerHTML += '<p class="warning">Warning:  
Very large ID. Decoding may be incomplete.</p>';
                }

                // Decode ID to string
                while (id > 0n) {
                    decodedString.push(_uniqueCharset[Number(id % BigInt(_uniqueCharset.length))]);
                    id = id / BigInt(_uniqueCharset.length);
                }

                if (decodedString.length === 0) {
                    throw new Error("Decoded string is empty.");
                }

                const decodedResult = decodedString.reverse().join('');
                document.getElementById('decoded-string-result').innerText =
"The decoded string is: " + decodedResult;

                // Update result display
                resultDisplay.innerHTML = `

                    <p>Decoding completed successfully.</p>
                    <p>Original ID: ${originalId.toString()}</p>
                    <p>Decoded string length: ${decodedResult.length}</p>
                    <p>Decoded string:</p>
                    <pre>${decodedResult}</pre>
                `;
            } catch (error) {
                resultDisplay.innerHTML = `<p class="error">Error decoding ID:  
${error.message}</p>`;
            }
        }, 100);
    } catch (error) {
        alert(`Error: ${error.message}`);
    }
}
```

```
/**  
 * Find optimal variable  
 */  
function findOptimalVariable() {  
    try {  
        const userNumber = BigInt(document.getElementById('user-number').value);  
        const outputFile = document.getElementById('output-file-optimal').value || 'optimal_variables.txt';  
  
        if (userNumber <= 0n) {  
            throw new Error("User number must be positive.");  
        }  
  
        const resultDisplay = document.getElementById('map-result-display');  
        resultDisplay.innerHTML = '<p>Finding optimal variables...</p>';  
  
        // Use setTimeout to allow UI to update before starting computation  
        setTimeout(() => {  
            try {  
                const charsetLength = BigInt(_uniqueCharset.length);  
                const extendedCharsetLength = BigInt(_uniqueCharset.length);  
                const generator = optimalVariableGenerator(userNumber,  
                    charsetLength, extendedCharsetLength);  
  
                const results = [];  
                for (let i = 0; i < 100; i++) {  
                    const optVar = generator.next().value;  
                    const decodedString = decodeIDFromNumber(optVar,  
                        _uniqueCharset);  
                    results.push({  
                        variable: optVar,  
                        string: decodedString,  
                        length: decodedString.length  
                    });  
                }  
  
                // Format the results for download  
                const resultString = results.map(r =>  
` ${r.variable}\t${r.string}`).join('\n');  
  
                // Download the file  
                downloadFile(outputFile, resultString);  
  
                // Display the results  
                let resultsHTML = '<p>Results (showing first 10):</p><table  
class="results-table">';  
                resultsHTML += '<tr><th>Variable</th><th>String</th>  
<th>Length</th></tr>';  
  
                results.slice(0, 10).forEach(r => {  
                    resultsHTML += `<tr><td>${r.variable}</td><td>${r.string}  
/<td>${r.length}</td></tr>`;  
                }  
            } catch (e) {  
                console.error(e);  
            }  
        } catch (e) {  
            console.error(e);  
        }  
    } catch (e) {  
        console.error(e);  
    }  
}
```

```
    });

    resultsHTML += '</table>';

    // Update result display
    resultDisplay.innerHTML = `
        <p>Successfully found 100 optimal variables.</p>
        <p>File saved as: ${outputFile}</p>
        ${resultsHTML}
    `;
}

} catch (error) {
    resultDisplay.innerHTML = `<p class="error">Error finding
optimal variables: ${error.message}</p>`;
}

}, 100);
} catch (error) {
    alert(`Error: ${error.message}`);
}
}

/***
 * Generate an optimal variable
 * @param {BigInt} userNumber - The user number
 * @param {BigInt} charsetLength - The charset length
 * @param {BigInt} extendedCharsetLength - The extended charset length
 * @returns {Object} Iterator for generating optimal variables
*/
function optimalVariableGenerator(userNumber, charsetLength,
extendedCharsetLength) {
    let k = 1n;
    return {
        next: function() {
            while (true) {
                const x = k * userNumber;
                if (x % extendedCharsetLength < charsetLength) {
                    const result = { value: x, done: false };
                    k += 1n;
                    return result;
                }
                k += 1n;
            }
        }
    };
}

/***
 * Check if a number is a perfect square
 * @param {number} n - Number to check
 * @returns {boolean} Whether the number is a perfect square
*/
function isPerfectSquare(n) {
    if (n <= 0) return false; // Grids must have a positive number of tiles
    const sqrt = Math.sqrt(n);
    return sqrt === Math.floor(sqrt);
```

```
}

/** 
 * Validate color indexes
 * @param {Array<number>} indexes - Array of color indexes
 * @returns {boolean} Whether the indexes are valid
 */
function areValidColorIndexes(indexes) {
    const maxColorIndex = Math.pow(16, 6); // Maximum valid color index is
16777216
    return indexes.every(index => index >= 1 && index <= maxColorIndex);
}

/** 
 * Update the color bar based on color indexes
 * @param {Array<number>} indexes - Array of color indexes
 */
function updateColorBar(indexes) {
    const colorBar = document.getElementById('color-bar');
    if (!colorBar) return;

    if (isPerfectSquare(indexes.length) && areValidColorIndexes(indexes)) {
        colorBar.style.backgroundColor = 'green'; // Valid grid
    } else {
        colorBar.style.backgroundColor = 'red'; // Invalid grid
    }
}

/** 
 * Decode an ID to color indexes
 * @param {string} idString - The ID string
 * @returns {Array<number>} Array of color indexes
 */
function decodeIDToColorIndexes(idString) {
    const indexes = [];
    const segmentLength = 7; // Each color index is represented by a 7-digit
segment

    // Split the integer string into 7-digit segments
    for (let i = 0; i < idString.length; i += segmentLength) {
        const segment = idString.slice(i, i + segmentLength);
        const index = parseInt(segment, 10);

        if (!isNaN(index)) {
            indexes.push(index);
        }
    }
    return indexes;
}

/** 
 * Display character and word count
 * @param {string} inputText - The input text
 */
```

```
function displayCharacterAndWordCount(inputText) {
    const characterCount = inputText.length;
    const wordCount = inputText.trim() === '' ? 0 :
inputText.trim().split(/\s+/).length;

    const charWordCountElement = document.getElementById('character-word-
count');
    if (charWordCountElement) {
        charWordCountElement.innerText = `Characters: ${characterCount},
Words: ${wordCount}`;
    }
}

/**
 * Decode an ID from a number
 * @param {BigInt} id - The ID to decode
 * @param {Array<string>} charset - The charset to use
 * @returns {string} The decoded string
 */
function decodeIDFromNumber(id, charset) {
    const decodedString = [];
    while (id > 0n) {
        decodedString.push(charset[Number(id % BigInt(charset.length))]);
        id = id / BigInt(charset.length);
    }
    return decodedString.reverse().join('');
}

/**
 * Download a file
 * @param {string} filename - The filename
 * @param {string} content - The file content
 */
function downloadFile(filename, content) {
    const blob = new Blob([content], { type: 'text/plain' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = filename;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
    URL.revokeObjectURL(url);
}

/**
 * Get the charset configuration
 * @returns {Array} The charset configuration
 */
function getCharsetConfig() {
    return [..._charsetConfig];
}

/**
```

```
* Get the unique charset
* @returns {Array<string>} The unique charset
*/
function getUniqueCharset() {
    return [..._uniqueCharset];
}

/***
 * Calculate a string ID (functional version for API access)
 * @param {string} inputString - The input string
 * @returns {BigInt} The calculated ID
*/
function calculateID(inputString) {
    if (!inputString || typeof inputString !== 'string') {
        throw new Error("Input must be a non-empty string");
    }

    let id = 0n;
    for (const char of inputString) {
        const charIndex = _uniqueCharset.indexOf(char);
        if (charIndex === -1) {
            throw new Error(`Character "${char}" is not in the charset`);
        }
        id = id * BigInt(_uniqueCharset.length) + BigInt(charIndex);
    }

    return id;
}

/***
 * Decode an ID to a string (functional version for API access)
 * @param {BigInt|string} id - The ID to decode
 * @returns {string} The decoded string
*/
function decodeIDToString(id) {
    try {
        id = BigInt(id);
    } catch (e) {
        throw new Error("Invalid ID format");
    }

    if (id < 0n) {
        throw new Error("ID cannot be negative");
    }

    const decodedString = [];
    while (id > 0n) {
        decodedString.push(_uniqueCharset[Number(id % BigInt(_uniqueCharset.length))]);
        id = id / BigInt(_uniqueCharset.length);
    }

    return decodedString.reverse().join('');
}
```

```

// Public API
return {
    initialize,
    loadMapSystem,
    showMapSection,
    calculateID,
    decodeIDtoString,
    getCharsetConfig,
    getUniqueCharset,
    isPerfectSquare,
    areValidColorIndexes,
    updateColorBar,
    decodeIDtoColorIndexes,
    downloadFile
};
})();
}

// Automatically initialize when the DOM is ready
document.addEventListener('DOMContentLoaded', function() {
    // Look for the main container
    const mainElement = document.querySelector('main');
    if (mainElement) {
        OperatorFramework.Map.initialize({
            containerId: mainElement.id || 'main'
        });
        console.log('Map System initialized');
    } else {
        console.warn('Main element not found, Map System not automatically initialized');
    }
});

```

## [98] integerDatabase/operator-persistence.js

- **Bytes:** 75169
- **Type:** text

```

/**
 * Operator Persistence System
 * A comprehensive state management and persistence system for the Operator OS
 *
 * This module handles saving and loading the state of all components:
 * - Data Structures (arrays, objects, sets, quadtrees)
 * - Algorithms (recursion, iteration, search, binary operations)
 * - System Operations (file system, memory management, process management)
 * - CRUD Database
 * - Advanced Concepts (serialization, compression, encoding, transactions)
 * - OperatorLang programming environment
 */

```

```
(function() {
    'use strict';

    // Configuration
    const CONFIG = {
        metadataKey: 'operatorSessionMetadata',
        chunkPrefix: 'operatorSessionChunk_',
        chunkSize: 1024 * 1024, // 1MB chunks
        version: '1.1',
        encryptionKey: 'OPERATOR_SYSTEM_ENCRYPTION_KEY',
        autoSaveKey: 'operatorAutoSave'
    };

    // State Registry - tracks all persistable components and their save/load
    // handlers
    const stateRegistry = new Map();

    // Initialize when document is ready
    document.addEventListener('DOMContentLoaded', function() {
        // Wait for all components to initialize
        setTimeout(initPersistenceSystem, 500);
    });

    /**
     * Initialize the persistence system
     */
    function initPersistenceSystem() {
        console.log("Initializing Operator Persistence System...");

        // Register all state handlers
        registerStateHandlers();

        // Create or enhance the persistence controls
        setupPersistenceControls();

        // Check for auto-recovery if configured
        checkAutoRecovery();

        // Register window unload handler to offer auto-save
        setupUnloadHandler();

        // Add this to the initPersistenceSystem function
        // Update workspace info every 30 seconds
        setInterval(updateWorkspaceInfo, 30000);
    }

    /**
     * Register state handlers for all components
     */
    function registerStateHandlers() {
        // Data Structures
        registerComponent('arrays', {
            selector: '#array-operations',
            save: saveArrayState,
        });
    }
})()
```

```
        load: loadArrayState
    });

registerComponent('objects', {
    selector: '#object-operations',
    save: saveObjectState,
    load: loadObjectState
});

registerComponent('sets', {
    selector: '#set-operations',
    save: saveSetState,
    load: loadSetState
});

registerComponent('quadtree', {
    selector: '#tree-operations',
    save: saveQuadtreeState,
    load: loadQuadtreeState
});

// CRUD Database
registerComponent('database', {
    selector: '#create-operations',
    save: saveDatabaseState,
    load: loadDatabaseState,
    priority: 10 // Higher priority components are processed first during
load
});

// System Operations
registerComponent('fileSystem', {
    selector: '#file-operations',
    save: saveFileSystemState,
    load: loadFileSystemState,
    priority: 20
});

registerComponent('memoryManagement', {
    selector: '#memory-operations',
    save: saveMemoryState,
    load: loadMemoryState
});

registerComponent('processManagement', {
    selector: '#process-operations',
    save: saveProcessState,
    load: loadProcessState
});

// Algorithm States
registerComponent('recursion', {
    selector: '#recursion-operations',
    save: saveRecursionState,
```

```
        load: loadRecursionState
    });

registerComponent('iteration', {
    selector: '#iteration-operations',
    save: saveIterationState,
    load: loadIterationState
});

registerComponent('search', {
    selector: '#search-operations',
    save: saveSearchState,
    load: loadSearchState
});

registerComponent('binaryOps', {
    selector: '#binary-operations',
    save: saveBinaryOperationsState,
    load: loadBinaryOperationsState
});

// Advanced Components
registerComponent('serialization', {
    selector: '#serialization-operations',
    save: saveSerializationState,
    load: loadSerializationState
});

registerComponent('compression', {
    selector: '#compression-operations',
    save: saveCompressionState,
    load: loadCompressionState
});

registerComponent('encoding', {
    selector: '#encoding-operations',
    save: saveEncodingState,
    load: loadEncodingState
});

registerComponent('transactions', {
    selector: '#transactions-operations',
    save: saveTransactionState,
    load: loadTransactionState
});

// Programming Language
registerComponent('operatorLang', {
    selector: '#programming-section',
    save: saveOperatorLangState,
    load: loadOperatorLangState,
    priority: 30
});
}
```

```
/**  
 * Register a component with the state registry  
 */  
function registerComponent(id, config) {  
    // Check if component exists in DOM  
    const element = document.querySelector(config.selector);  
    if (!element) {  
        console.log(`Component ${id} not found in DOM, skipping registration`);  
        return;  
    }  
  
    // Set default priority (0) if not specified  
    if (config.priority === undefined) {  
        config.priority = 0;  
    }  
  
    // Register component  
    stateRegistry.set(id, {  
        id: id,  
        element: element,  
        save: config.save,  
        load: config.load,  
        priority: config.priority,  
        active: true  
    });  
  
    console.log(`Registered component: ${id}`);  
}  
  
function setupPersistenceControls() {  
    // Check for existing controls in the main navigation  
    const mainNav = document.getElementById('main-nav');  
    if (mainNav) {  
        // Create session section in nav if it doesn't exist  
        // This line has the error:  
        // let sessionSection = mainNav.querySelector('.nav-section:has(h3:contains("Session"))');  
  
        // Find Session section without using :has or :contains  
        let sessionSection = null;  
        const navSections = mainNav.querySelectorAll('.nav-section');  
        for (const section of navSections) {  
            const h3 = section.querySelector('h3');  
            if (h3 && h3.textContent === 'Session') {  
                sessionSection = section;  
                break;  
            }  
        }  
  
        if (!sessionSection) {  
            sessionSection = document.createElement('div');  
            sessionSection.className = 'nav-section';  
        }  
    }  
}
```

```
sessionSection.innerHTML = `<h3>Session</h3><div class="persistence-controls"><button id="save-session" title="Save current workspace">Save Workspace</button><button id="load-session" title="Load saved workspace">Load Workspace</button><button id="auto-save-toggle" class="button-secondary" title="Toggle automatic saving">Auto Save: <span id="auto-save-status">Off</span></button></div>`;  
mainNav.appendChild(sessionSection);  
  
// Set up event listeners  
const saveButton = document.getElementById('save-session');  
const loadButton = document.getElementById('load-session');  
const autoSaveToggle = document.getElementById('auto-save-toggle');  
  
if (saveButton) {  
    saveButton.removeEventListener('click', saveAllStates);  
    saveButton.addEventListener('click', saveAllStates);  
}  
  
if (loadButton) {  
    loadButton.removeEventListener('click', loadWorkspaceFromFile);  
    loadButton.addEventListener('click', loadWorkspaceFromFile);  
}  
  
if (autoSaveToggle) {  
    const autoSaveStatus = document.getElementById('auto-save-status');  
    const autoSaveEnabled = localStorage.getItem(CONFIG.autoSaveKey)  
==== 'true';  
  
    // Set initial status  
    if (autoSaveStatus) {  
        autoSaveStatus.textContent = autoSaveEnabled ? 'On' : 'Off';  
    }  
  
    // Toggle auto-save on click  
    autoSaveToggle.addEventListener('click', function() {  
        const currentState = localStorage.getItem(CONFIG.autoSaveKey)  
==== 'true';  
        const newStatus = !currentState;  
  
        localStorage.setItem(CONFIG.autoSaveKey,  
newStatus.toString());  
  
        if (autoSaveStatus) {  
            autoSaveStatus.textContent = newStatus ? 'On' : 'Off';  
        }  
    });  
}
```

```
        showNotification(`Auto save ${newStatus ? 'enabled' :
'disabled'}}, 'info');
    });
}

console.log("Set up persistence controls in main navigation");
} else {
    // Create floating controls if main nav not found
    createFloatingPersistenceControls();
}

// Add this to operator-persistence.js inside the setupPersistenceControls
function createSessionManagementSection() {
    // Check if the section already exists
    if (document.getElementById('session-management')) {
        return;
    }

    // Create a new section
    const section = document.createElement('section');
    section.id = 'session-management';
    section.className = 'operation-section';

    // Add the section content
    section.innerHTML =
        `

## Session Management


        <div class="operation-container">
            <div class="operation-controls">
                <h3>Workspace Management</h3>
                <div class="control-group">
                    <button id="save-workspace-btn" class="button">Save
                    Workspace</button>
                    <button id="load-workspace-btn" class="button">Load
                    Workspace</button>
                </div>
                <div class="control-group mt-20">
                    <h4>Backup Options</h4>
                    <button id="export-workspace-btn"
class="button">Export Workspace</button>
                    <button id="import-workspace-btn"
class="button">Import Workspace</button>
                </div>
                <div class="control-group mt-20">
                    <h4>Auto-Save</h4>
                    <div class="toggle-container">
                        <label class="toggle">
                            <input type="checkbox" id="auto-save-toggle">
                            <span class="toggle-slider"></span>
                        </label>
                        <span id="auto-save-label">Auto-save is off</span>
                    </div>
                </div>
            </div>
        </div>

```

```
        <p class="hint-text">When enabled, your workspace will  
be automatically saved when you leave the page.</p>  
        </div>  
    </div>  
    <div class="operation-result">  
        <h3>Workspace Status</h3>  
        <div id="workspace-info" class="workspace-info">  
            <div class="info-item">  
                <span class="info-label">Last Saved:</span>  
                <span id="last-saved-time">Never</span>  
            </div>  
            <div class="info-item">  
                <span class="info-label">Components:</span>  
                <span id="component-count">0</span>  
            </div>  
            <div class="info-item">  
                <span class="info-label">Storage Used:</span>  
                <span id="storage-used">0 KB</span>  
            </div>  
        </div>  
        <div class="mt-20">  
            <h4>Recent Actions</h4>  
            <div id="session-history" class="session-history">  
                <div class="empty-history">No recent actions</div>  
            </div>  
        </div>  
        <div class="mt-20">  
            <button id="clear-storage-btn" class="button button-secondary">Clear Stored Data</button>  
        </div>  
    </div>  
`;  
  
// Add the section to the main content  
const main = document.querySelector('main');  
if (main) {  
    main.appendChild(section);  
  
    // Add to navigation  
    const navSection = document.querySelector('#main-nav');  
    if (navSection) {  
        const sessionNav = document.createElement('div');  
        sessionNav.className = 'nav-section';  
        sessionNav.innerHTML = `<h3>Session</h3>  
        <button data-section="session-management">Workspace  
Management</button>`;  
        navSection.appendChild(sessionNav);  
    }  
  
    // Set up event listeners  
    setupSessionManagementListeners();
```

```
        updateWorkspaceInfo();
    }

}

// Call this function from setupPersistenceControls
createSessionManagementSection();

}

/** 
 * Create floating persistence controls if they don't exist in main nav
 */
function createFloatingPersistenceControls() {
    // Check if floating controls already exist
    if (document.getElementById('floating-persistence-controls')) {
        return;
    }

    // Create container
    const controlsContainer = document.createElement('div');
    controlsContainer.id = 'floating-persistence-controls';
    controlsContainer.className = 'persistence-controls-container';
    controlsContainer.style.cssText = `
        position: fixed;
        bottom: 20px;
        right: 20px;
        background-color: white;
        border: 1px solid var(--secondary-color);
        border-radius: 5px;
        padding: 10px;
        box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        z-index: 1000;
        display: flex;
        flex-direction: column;
        gap: 10px;
    `;

    // Add buttons
    const saveButton = document.createElement('button');
    saveButton.textContent = 'Save Workspace';
    saveButton.className = 'button';
    saveButton.addEventListener('click', saveAllStates);

    const loadButton = document.createElement('button');
    loadButton.textContent = 'Load Workspace';
    loadButton.className = 'button';
    loadButton.addEventListener('click', loadWorkspaceFromFile);

    const autoSaveToggle = document.createElement('button');
    autoSaveToggle.className = 'button button-secondary';

    const autoSaveEnabled = localStorage.getItem(CONFIG.autoSaveKey) ===
'true';
    autoSaveToggle.textContent = `Auto Save: ${autoSaveEnabled ? 'On' :
```

```
'Off'}';

    autoSaveToggle.addEventListener('click', function() {
        const currentStatus = localStorage.getItem(CONFIG.autoSaveKey) ===
        'true';
        const newStatus = !currentStatus;

        localStorage.setItem(CONFIG.autoSaveKey, newStatus.toString());
        autoSaveToggle.textContent = `Auto Save: ${newStatus ? 'On' : 'Off'}`;

        showNotification(`Auto save ${newStatus ? 'enabled' : 'disabled'}`,
        'info');
    });

    // Add to container
    controlsContainer.appendChild(saveButton);
    controlsContainer.appendChild(loadButton);
    controlsContainer.appendChild(autoSaveToggle);

    // Add to document
    document.body.appendChild(controlsContainer);
    console.log("Created floating persistence controls");
}

/***
 * Save all component states
 */
async function saveAllStates() {
    try {
        console.log("Saving all states...");
        showNotification('Saving workspace...', 'info');

        // Collect state from all registered components
        const fullState = {
            version: CONFIG.version,
            timestamp: new Date().toISOString(),
            components: {}
        };

        // Save each component's state
        for (const [id, component] of stateRegistry.entries()) {
            if (!component.active) continue;

            try {
                const componentState = await component.save();
                if (componentState !== undefined) {
                    fullState.components[id] = componentState;
                }
            } catch (error) {
                console.error(`Error saving state for ${id}:`, error);
            }
        }

        // Convert to JSON and encrypt
    }
}
```

```
const jsonData = JSON.stringify(fullState);
const encryptedData = encryptData(jsonData);

// Split into chunks for localStorage
const chunks = chunkString(encryptedData, CONFIG.chunkSize);

// Store metadata
localStorage.setItem(CONFIG.metadataKey, JSON.stringify({
    version: CONFIG.version,
    timestamp: new Date().toISOString(),
    chunks: chunks.length,
    size: encryptedData.length
}));

// Store chunks
chunks.forEach((chunk, index) => {
    localStorage.setItem(`#${CONFIG.chunkPrefix}${index}`, chunk);
});

// Create download link for backup
offerDownload(fullState);

showNotification('Workspace saved successfully', 'success');
// Add to the end of the saveAllStates function, before the return
statement:
addHistoryItem('Workspace saved');
updateWorkspaceInfo();
} catch (error) {
    console.error('Error saving workspace:', error);
    showNotification('Failed to save workspace: ' + error.message,
'error');
}
}

/**
 * Load workspace from a file selected by the user
 */
function loadWorkspaceFromFile() {
    // Create file input for loading JSON
    const fileInput = document.createElement('input');
    fileInput.type = 'file';
    fileInput.accept = '.json';
    fileInput.style.display = 'none';
    document.body.appendChild(fileInput);

    fileInput.addEventListener('change', function(e) {
        const file = e.target.files[0];
        if (!file) return;

        const reader = new FileReader();
        reader.onload = function(e) {
            try {
                // Parse the file content
                const workspaceData = JSON.parse(e.target.result);

```

```
// Basic validation
if (!workspaceData.timestamp) {
    throw new Error('Invalid workspace file format');
}

// Confirm before loading
if (!confirm(`Load workspace from ${new Date(workspaceData.timestamp).toLocaleString()}? This will replace your current workspace.`)) {
    return;
}

// Load the workspace
loadFromObject(workspaceData);
} catch (error) {
    console.error('Error loading workspace file:', error);
    showNotification('Failed to load workspace: ' + error.message,
'error');
}
};

reader.onerror = function() {
    showNotification('Error reading file', 'error');
};

reader.readAsText(file);

// Clean up
document.body.removeChild(fileInput);
});

// Trigger file selection
fileInput.click();
}

/**
 * Load workspace from localStorage
 */
async function loadFromLocalStorage() {
    try {
        // Check if there's a saved state
        const metadataStr = localStorage.getItem(CONFIG.metadataKey);
        if (!metadataStr) {
            showNotification('No saved workspace found in browser storage',
'error');
            return;
        }

        showNotification('Loading workspace...', 'info');

        // Parse metadata
        const metadata = JSON.parse(metadataStr);
```

```
// Retrieve and combine chunks
let encryptedData = '';
for (let i = 0; i < metadata.chunks; i++) {
    const chunk = localStorage.getItem(`#${CONFIG.chunkPrefix}${i}`);
    if (!chunk) {
        throw new Error(`Missing session data chunk ${i}`);
    }
    encryptedData += chunk;
}

// Decrypt data
const jsonData = decryptData(encryptedData);
const fullState = JSON.parse(jsonData);

// Load the workspace
await loadFromObject(fullState);
} catch (error) {
    console.error('Error loading from localStorage:', error);
    showNotification('Failed to load workspace: ' + error.message,
'error');
}
}

/**
 * Load state from a workspace object
 */
async function loadFromObject(workspaceData) {
    try {
        showNotification('Loading workspace...', 'info');

        // Verify version compatibility
        if (workspaceData.version && workspaceData.version !== CONFIG.version)
{
            console.warn(`Version mismatch: saved=${workspaceData.version},
current=${CONFIG.version}`);
            // Allow loading but warn
            showNotification('Loading workspace from a different version',
'warning');
        }
    }

    // Handle both formats:
    // 1. New format: { components: { id: state } }
    // 2. Legacy format: { virtualFS, database, etc. }
    const hasComponents = workspaceData.components && typeof
workspaceData.components === 'object';

    if (hasComponents) {
        // Get components sorted by priority
        const sortedComponents = Array.from(stateRegistry.values())
            .sort((a, b) => b.priority - a.priority);

        // Load each component state
        for (const component of sortedComponents) {
            if (!component.active) continue;
    }
}
```

```
const id = component.id;
if (workspaceData.components[id]) {
    try {
        await component.load(workspaceData.components[id]);
        console.log(`Loaded state for ${id}`);
    } catch (error) {
        console.error(`Error loading state for ${id}:`, error);
    }
}
} else {
    // Legacy format - map directly to components
    // File System
    if (workspaceData.virtualFS) {
        try {
            window.virtualFS = workspaceData.virtualFS;
            if (typeof window.renderFileTree === 'function') {
                window.renderFileTree();
            }
        } catch (error) {
            console.error('Error loading file system:', error);
        }
    }
}

// Database
if (workspaceData.database && workspaceData.database.records) {
    try {
        if (window.database) {
            window.database.deleteAll();
            workspaceData.database.records.forEach(([id, record]) => {
                window.database.records.set(id, record);
            });

            if (window.databaseDisplayUpdater) {

                window.databaseDisplayUpdater.updateDatabaseDisplay();
            }
        }
    } catch (error) {
        console.error('Error loading database:', error);
    }
}

// Memory
if (workspaceData.memorySlots && window.memoryManager) {
    try {
        window.memoryManager.memorySlots =
workspaceData.memorySlots;
        if (typeof window.updateMemoryDisplay === 'function') {
            window.updateMemoryDisplay();
        }
    }
}
```

```
        } catch (error) {
            console.error('Error loading memory state:', error);
        }
    }

    // OperatorLang
    if (workspaceData.programming && window.operatorLang) {
        try {
            const codeEditor = document.getElementById('code-editor');
            if (codeEditor && workspaceData.programming.code) {
                codeEditor.value = workspaceData.programming.code;
            }

            if (workspaceData.programming.variables) {
                window.operatorLang.variables = new
Map(workspaceData.programming.variables);
            }
        } catch (error) {
            console.error('Error loading programming state:', error);
        }
    }
    // Update workspace info display
    updateWorkspaceInfo();
}

showNotification('Workspace loaded successfully', 'success');
} catch (error) {
    console.error('Error loading workspace:', error);
    showNotification('Failed to load workspace: ' + error.message,
'error');
}
}

/** 
 * Offer to download state as a backup file
 */
function offerDownload(state) {
    try {
        // Create a JSON file
        const blob = new Blob([JSON.stringify(state)], {type:
'application/json'});
        const url = URL.createObjectURL(blob);

        // Create a temporary link element
        const a = document.createElement('a');
        a.href = url;
        a.download = `operator_workspace_${new
Date().toISOString().replace(/:/g, '-')}.json`;

        // Ask user if they want to download
        if (confirm('Would you like to download a backup of your workspace?'))
{
            a.click();
        }
    }
}
```

```
// Clean up
setTimeout(() => URL.revokeObjectURL(url), 5000);
} catch (error) {
  console.error('Error creating download:', error);
}
}

// Modify checkAutoRecovery to update UI:
function checkAutoRecovery() {
  // Check for a saved state
  const metadataStr = localStorage.getItem(CONFIG.metadataKey);
  if (!metadataStr) return;

  try {
    const metadata = JSON.parse(metadataStr);
    const savedDate = new Date(metadata.timestamp);
    const now = new Date();

    // Update the workspace info display
    updateWorkspaceInfo();

    // If the saved state is recent (within the last day)
    if ((now - savedDate) < 24 * 60 * 60 * 1000) {
      // Show recovery banner
      const banner = document.createElement('div');
      banner.className = 'recovery-banner';
      banner.style.cssText =
        `position: fixed;
         top: 0;
         left: 0;
         right: 0;
         background-color: var(--primary-color);
         color: white;
         padding: 10px 20px;
         text-align: center;
         z-index: 2000;
         display: flex;
         justify-content: center;
         align-items: center;
         gap: 20px;
        `;
      banner.innerHTML =
        `

Saved workspace found from ${savedDate.toLocaleString()}. Would you like to restore it?


        <div>
          <button id="restore-workspace" class="button">Restore</button>
          <button id="dismiss-banner" class="button button-secondary">Dismiss</button>
        </div>
      `;
    }
  }
}
```

```
        document.body.appendChild(banner);

        // Add event listeners
        document.getElementById('restore-
workspace').addEventListener('click', () => {
            loadFromLocalStorage();
            document.body.removeChild(banner);
            addHistoryItem('Workspace restored from auto-recovery');
        });

        document.getElementById('dismiss-
banner').addEventListener('click', () => {
            document.body.removeChild(banner);
        });
    }

} catch (error) {
    console.error('Error checking for recovery:', error);
}

}

/***
 * Setup window unload handler
 */
function setupUnloadHandler() {
    window.addEventListener('beforeunload', function(e) {
        // Check if auto-save is enabled
        const autoSave = localStorage.getItem(CONFIG.autoSaveKey) === 'true';
        if (autoSave) {
            // Attempt to save state (async, might not complete)
            saveAllStates();
            return;
        }

        // Otherwise, show confirmation dialog
        const confirmationMessage = 'You have unsaved changes. Would you like
to save your workspace before leaving?';
        e.returnValue = confirmationMessage;
        return confirmationMessage;
    });
}

/***
 * Show a notification to the user
 */
function showNotification(message, type = 'info') {
    // Check if a notification container exists
    let container = document.getElementById('persistence-notifications');
    if (!container) {
        // Create a container
        container = document.createElement('div');
        container.id = 'persistence-notifications';
        container.style.cssText =
            `position: fixed;
            bottom: 20px;
            width: fit-content;
            background-color: white;
            border: 1px solid #ccc;
            padding: 10px;
            font-family: sans-serif;
            font-size: 14px;
            color: black;
            z-index: 1000;
            opacity: 0;
            transition: opacity 0.3s ease-in-out;
            pointer-events: none;
            visibility: hidden;
            transform: translateY(-50%);`;
```

```
        left: 20px;
        z-index: 2000;
        display: flex;
        flex-direction: column;
        gap: 10px;
    `;
    document.body.appendChild(container);
}

// Create notification
const notification = document.createElement('div');
notification.className = `persistence-notification ${type}`;
notification.style.cssText =
    background-color: white;
    border-left: 4px solid ${getColor(type)};
    padding: 15px 20px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    border-radius: 4px;
    max-width: 300px;
    animation: fadeIn 0.3s ease-out;
    position: relative;
`;
notification.innerHTML =
<div style="margin-right: 20px;">${message}</div>
<button class="close-notification" style="
    position: absolute;
    top: 5px;
    right: 5px;
    border: none;
    background: none;
    cursor: pointer;
    font-size: 16px;
    color: #999;
">&times;</button>
`;
// Add to container
container.appendChild(notification);

// Add close button event
notification.querySelector('.close-notification').addEventListener('click', () => {
    container.removeChild(notification);
});

// Auto-hide after a delay
setTimeout(() => {
    notification.style.opacity = '0';
    notification.style.transition = 'opacity 0.3s';
    setTimeout(() => {
        if (container.contains(notification)) {
            container.removeChild(notification);
        }
    })
})
```

```
        }, 300);
    }, 5000);
}

/** 
 * Get color for notification type
 */
function getTypeColor(type) {
    switch (type) {
        case 'success': return '#4caf50';
        case 'error': return '#f44336';
        case 'warning': return '#ff9800';
        case 'info':
        default: return '#2196f3';
    }
}

/** 
 * Encryption functions for secure storage
 */
function encryptData(data) {
    try {
        // Simple XOR encryption for demonstration
        // In production, use a proper encryption library
        const key = CONFIG.encryptionKey;
        let result = '';

        for (let i = 0; i < data.length; i++) {
            const charCode = data.charCodeAt(i) ^ key.charCodeAt(i % key.length);
            result += String.fromCharCode(charCode);
        }

        // Convert to base64 for safer storage
        return btoa(result);
    } catch (error) {
        console.error('Error encrypting data:', error);
        return btoa(data); // Fallback to basic encoding
    }
}

function decryptData(encrypted) {
    try {
        // Decode from base64
        const encoded = atob(encrypted);
        const key = CONFIG.encryptionKey;
        let result = '';

        for (let i = 0; i < encoded.length; i++) {
            const charCode = encoded.charCodeAt(i) ^ key.charCodeAt(i % key.length);
            result += String.fromCharCode(charCode);
        }
    }
}
```

```
        return result;
    } catch (error) {
        console.error('Error decrypting data:', error);
        return atob(encrypted); // Fallback to basic decoding
    }
}

/***
 * Split a string into chunks
 */
function chunkString(str, size) {
    const chunks = [];
    for (let i = 0; i < str.length; i += size) {
        chunks.push(str.substring(i, i + size));
    }
    return chunks;
}

/***
 * Component State Handlers
 * These functions handle saving and loading state for specific components
*/
// ----- Data Structures -----

async function saveArrayState() {
    const container = document.getElementById('array-operations');
    if (!container) return null;

    const inputElement = container.querySelector('#array-input');
    const resultContainer = container.querySelector('#array-result');

    return {
        input: inputElement ? inputElement.value : '',
        // Also capture the current visualization state if possible
        displayHTML: resultContainer ? resultContainer.innerHTML : ''
    };
}

async function loadArrayState(state) {
    const container = document.getElementById('array-operations');
    if (!container || !state) return;

    const inputElement = container.querySelector('#array-input');
    if (inputElement && state.input) {
        inputElement.value = state.input;
    }

    // Attempt to restore visualization
    const resultContainer = container.querySelector('#array-result');
    if (resultContainer && state.displayHTML) {
        resultContainer.innerHTML = state.displayHTML;
    }
}
```

```
async function saveObjectState() {
    const container = document.getElementById('object-operations');
    if (!container) return null;

    // Try to access the current object state
    let currentState = {};
    try {
        currentState = window.currentObject || {};
    } catch (e) {
        console.warn("Couldn't access current object state");
    }

    return {
        currentState: currentState,
        displayHTML: container.querySelector('#object-result')?.innerHTML
    };
}

async function loadObjectState(state) {
    const container = document.getElementById('object-operations');
    if (!container || !state) return;

    // Try to restore the object state
    try {
        window.currentObject = state.currentState || {};
    } catch (e) {
        console.warn("Couldn't restore object state");
    }

    // Restore display
    if (state.displayHTML) {
        const resultContainer = container.querySelector('#object-result');
        if (resultContainer) {
            resultContainer.innerHTML = state.displayHTML;
        }
    }
}

async function saveSetState() {
    const container = document.getElementById('set-operations');
    if (!container) return null;

    const setAInput = container.querySelector('#set-a-input');
    const setBInput = container.querySelector('#set-b-input');

    return {
        setA: setAInput ? setAInput.value : '',
        setB: setBInput ? setBInput.value : '',
        displayHTML: container.querySelector('#set-result')?.innerHTML
    };
}

async function loadSetState(state) {
```

```
const container = document.getElementById('set-operations');
if (!container || !state) return;

if (state.setA) {
    const input = container.querySelector('#set-a-input');
    if (input) input.value = state.setA;
}

if (state.setB) {
    const input = container.querySelector('#set-b-input');
    if (input) input.value = state.setB;
}

if (state.displayHTML) {
    const result = container.querySelector('#set-result');
    if (result) result.innerHTML = state.displayHTML;
}
}

async function saveQuadtreeState() {
    const container = document.getElementById('tree-operations');
    if (!container) return null;

    // Get quadtree parameters
    const maxDepth = document.getElementById('max-depth')?.value || 3;
    const quadtreeSize = document.getElementById('quadtree-size')?.value || 400;

    // Get the HTML structure of all cells
    const quadtreeContainer = document.getElementById('quadtree-container');
    const cellsHTML = quadtreeContainer?.innerHTML || '';

    // Try to find customized cells
    const customizedCells = [];
    quadtreeContainer?.querySelectorAll('.quadtree-cell').forEach(cell => {
        if (cell.classList.contains('customized') ||
            cell.style.backgroundColor ||
            cell.querySelector('.quadtree-cell-custom-text') ||
            cell.querySelector('.quadtree-cell-image')) {

            // Extract the customization details
            const customization = {
                x: parseFloat(cell.style.left) || 0,
                y: parseFloat(cell.style.top) || 0,
                width: parseFloat(cell.style.width) || 0,
                height: parseFloat(cell.style.height) || 0,
                backgroundColor: cell.style.backgroundColor,
                opacity: cell.style.opacity,
                customized: cell.classList.contains('customized')
            };

            // Extract text if present
            const textElement = cell.querySelector('.quadtree-cell-custom-
text');
        }
    });
}
```

```
        if (textElement) {
            customization.text = {
                content: textElement.textContent,
                color: textElement.style.color,
                fontSize: textElement.style.fontSize
            };
        }

        // Extract image if present
        const imageElement = cell.querySelector('.quadtree-cell-image');
        if (imageElement) {
            customization.image = {
                src: imageElement.src
            };
        }

        customizedCells.push(customization);
    }
});

return {
    maxDepth,
    quadtreeSize,
    cellsHTML,
    customizedCells
};
}

async function loadQuadtreeState(state) {
    const container = document.getElementById('tree-operations');
    if (!container || !state) return;

    // Set quadtree parameters
    if (state.maxDepth) {
        const input = document.getElementById('max-depth');
        if (input) input.value = state.maxDepth;
    }

    if (state.quadtreeSize) {
        const input = document.getElementById('quadtree-size');
        if (input) input.value = state.quadtreeSize;
    }

    // Recreate the quadtree
    const createBtn = document.getElementById('create-quadtree');
    if (createBtn) {
        createBtn.click();

        // Wait for the quadtree to be created
        setTimeout(() => {
            // Try to restore customized cells
            const quadtreeContainer = document.getElementById('quadtree-
container');
            if (!quadtreeContainer || !state.customizedCells) return;
        });
    }
}
```

```
state.customizedCells.forEach(customization => {
    // Find a cell at the same position
    const cells = quadtreeContainer.querySelectorAll('.quadtree-cell');
    let matchingCell = null;

    for (const cell of cells) {
        const x = parseFloat(cell.style.left) || 0;
        const y = parseFloat(cell.style.top) || 0;
        const width = parseFloat(cell.style.width) || 0;
        const height = parseFloat(cell.style.height) || 0;

        if (Math.abs(x - customization.x) < 1 &&
            Math.abs(y - customization.y) < 1 &&
            Math.abs(width - customization.width) < 1 &&
            Math.abs(height - customization.height) < 1) {
            matchingCell = cell;
            break;
        }
    }

    if (matchingCell) {
        // Apply customizations
        if (customization.backgroundColor) {
            matchingCell.style.backgroundColor =
customization.backgroundColor;
        }

        if (customization.opacity) {
            matchingCell.style.opacity = customization.opacity;
        }

        if (customization.customized) {
            matchingCell.classList.add('customized');
        }

        // Add text if present
        if (customization.text) {
            const textElement = document.createElement('div');
            textElement.className = 'quadtree-cell-custom-text';
            textElement.textContent = customization.text.content;
            textElement.style.color = customization.text.color;
            textElement.style.fontSize =
customization.text.fontSize;
            matchingCell.appendChild(textElement);
        }

        // Add image if present
        if (customization.image && customization.image.src) {
            const imageElement = document.createElement('img');
            imageElement.className = 'quadtree-cell-image';
            imageElement.src = customization.image.src;
            matchingCell.appendChild(imageElement);
        }
    }
})
```

```
        }
    }
}, 500);
}
}

// ----- CRUD Database -----

async function saveDatabaseState() {
    if (!window.database) return null;

    try {
        // Capture database records
        const records =
Array.from(window.database.records.entries()).map(([id, record]) => {
            // Clone the record to avoid circular references
            return {
                id: id,
                name: record.name,
                data: record.data,
                createdAt: record.createdAt ? record.createdAt.toISOString() :
null,
                updatedAt: record.updatedAt ? record.updatedAt.toISOString() :
null
            };
        });
    }

        return { records };
    } catch (e) {
        console.error('Error saving database state:', e);
        return null;
    }
}

async function loadDatabaseState(state) {
    if (!window.database || !state || !state.records) return;

    try {
        // Clear database
        window.database.deleteAll();

        // Restore records
        state.records.forEach(record => {
            window.database.create({
                id: record.id,
                name: record.name,
                data: record.data,
                createdAt: record.createdAt ? new Date(record.createdAt) : new
Date(),
                updatedAt: record.updatedAt ? new Date(record.updatedAt) :
null
            });
        });
    }
}
```

```
// Update display if possible
if (window.databaseDisplayUpdater &&
window.databaseDisplayUpdater.updateDatabaseDisplay) {
    window.databaseDisplayUpdater.updateDatabaseDisplay();
}
} catch (e) {
    console.error('Error loading database state:', e);
}
}

// ----- System Operations -----


async function saveFileSystemState() {
try {
    // Get the virtual file system or create an empty one
    const virtualFS = window.virtualFS || {
        root: { type: 'directory', name: 'root', children: {} }
    };

    return virtualFS;
} catch (e) {
    console.error('Error saving file system state:', e);
    return null;
}
}

async function loadFileSystemState(state) {
if (!state) return;

try {
    // Restore virtual file system
    window.virtualFS = state;

    // Update file tree display if possible
    if (typeof window.renderFileTree === 'function') {
        window.renderFileTree();
    }
} catch (e) {
    console.error('Error loading file system state:', e);
}
}

async function saveMemoryState() {
try {
    if (!window.memoryManager) return null;

    return {
        memorySlots: window.memoryManager.memorySlots
    };
} catch (e) {
    console.error('Error saving memory state:', e);
    return null;
}
}
```

```
}

async function loadMemoryState(state) {
    if (!window.memoryManager || !state || !state.memorySlots) return;

    try {
        // Restore memory slots
        window.memoryManager.memorySlots = state.memorySlots;

        // Update memory display if possible
        if (typeof window.updateMemoryDisplay === 'function') {
            window.updateMemoryDisplay();
        }
    } catch (e) {
        console.error('Error loading memory state:', e);
    }
}

async function saveProcessState() {
    try {
        // Process manager is mostly runtime state, not much to save
        return {
            // Could save process templates/configurations if needed
        };
    } catch (e) {
        console.error('Error saving process state:', e);
        return null;
    }
}

async function loadProcessState(state) {
    // Mainly for future expansion
    if (!state) return;

    try {
        // Restore any saved process configurations
    } catch (e) {
        console.error('Error loading process state:', e);
    }
}

// ----- Algorithm States -----

async function saveRecursionState() {
    const container = document.getElementById('recursion-operations');
    if (!container) return null;

    return {
        n: document.getElementById('recursion-n')?.value,
        result: document.getElementById('recursion-result')?.innerHTML
    };
}

async function loadRecursionState(state) {
```

```
if (!state) return;

try {
    if (state.n) {
        const input = document.getElementById('recursion-n');
        if (input) input.value = state.n;
    }

    if (state.result) {
        const result = document.getElementById('recursion-result');
        if (result) result.innerHTML = state.result;
    }
} catch (e) {
    console.error('Error loading recursion state:', e);
}

}

async function saveIterationState() {
    const container = document.getElementById('iteration-operations');
    if (!container) return null;

    return {
        n: document.getElementById('iteration-n')?.value,
        charset: document.getElementById('charset-input')?.value,
        result: document.getElementById('iteration-result')?.innerHTML
    };
}

async function loadIterationState(state) {
    if (!state) return;

    try {
        if (state.n) {
            const input = document.getElementById('iteration-n');
            if (input) input.value = state.n;
        }

        if (state.charset) {
            const input = document.getElementById('charset-input');
            if (input) input.value = state.charset;
        }

        if (state.result) {
            const result = document.getElementById('iteration-result');
            if (result) result.innerHTML = state.result;
        }
    } catch (e) {
        console.error('Error loading iteration state:', e);
    }
}

async function saveSearchState() {
    const container = document.getElementById('search-operations');
    if (!container) return null;
```

```
        return {
            data: document.getElementById('search-data')?.value,
            query: document.getElementById('search-query')?.value,
            result: document.getElementById('search-result')?.innerHTML
        };
    }

    async function loadSearchState(state) {
        if (!state) return;

        try {
            if (state.data) {
                const input = document.getElementById('search-data');
                if (input) input.value = state.data;
            }

            if (state.query) {
                const input = document.getElementById('search-query');
                if (input) input.value = state.query;
            }

            if (state.result) {
                const result = document.getElementById('search-result');
                if (result) result.innerHTML = state.result;
            }
        } catch (e) {
            console.error('Error loading search state:', e);
        }
    }

    async function saveBinaryOperationsState() {
        const container = document.getElementById('binary-operations');
        if (!container) return null;

        return {
            valueA: document.getElementById('binary-input-a')?.value,
            valueB: document.getElementById('binary-input-b')?.value,
            result: document.getElementById('binary-result')?.innerHTML
        };
    }

    async function loadBinaryOperationsState(state) {
        if (!state) return;

        try {
            if (state.valueA) {
                const input = document.getElementById('binary-input-a');
                if (input) input.value = state.valueA;
            }

            if (state.valueB) {
                const input = document.getElementById('binary-input-b');
                if (input) input.value = state.valueB;
            }
        }
    }
}
```

```
        }

        if (state.result) {
            const result = document.getElementById('binary-result');
            if (result) result.innerHTML = state.result;
        }
    } catch (e) {
    console.error('Error loading binary operations state:', e);
}
}

// ----- Advanced Components -----


async function saveSerializationState() {
    const container = document.getElementById('serialization-operations');
    if (!container) return null;

    return {
        input: document.getElementById('serialize-input')?.value,
        format: document.getElementById('serialization-format')?.value,
        result: document.getElementById('serialization-result')?.innerHTML
    };
}

async function loadSerializationState(state) {
    if (!state) return;

    try {
        if (state.input) {
            const input = document.getElementById('serialize-input');
            if (input) input.value = state.input;
        }

        if (state.format) {
            const select = document.getElementById('serialization-format');
            if (select) select.value = state.format;
        }

        if (state.result) {
            const result = document.getElementById('serialization-result');
            if (result) result.innerHTML = state.result;
        }
    } catch (e) {
    console.error('Error loading serialization state:', e);
}
}

async function saveCompressionState() {
    const container = document.getElementById('compression-operations');
    if (!container) return null;

    return {
        input: document.getElementById('compression-input')?.value,
        method: document.getElementById('compression-method')?.value,
    };
}
```

```
        result: document.getElementById('compression-result')?.innerHTML,
        stats: document.getElementById('compression-stats')?.innerHTML
    );
}

async function loadCompressionState(state) {
    if (!state) return;

    try {
        if (state.input) {
            const input = document.getElementById('compression-input');
            if (input) input.value = state.input;
        }

        if (state.method) {
            const select = document.getElementById('compression-method');
            if (select) select.value = state.method;
        }

        if (state.result) {
            const result = document.getElementById('compression-result');
            if (result) result.innerHTML = state.result;
        }

        if (state.stats) {
            const stats = document.getElementById('compression-stats');
            if (stats) stats.innerHTML = state.stats;
        }
    } catch (e) {
        console.error('Error loading compression state:', e);
    }
}

async function saveEncodingState() {
    const container = document.getElementById('encoding-operations');
    if (!container) return null;

    return {
        input: document.getElementById('encoding-input')?.value,
        method: document.getElementById('encoding-method')?.value,
        result: document.getElementById('encoding-result')?.innerHTML
    };
}

async function loadEncodingState(state) {
    if (!state) return;

    try {
        if (state.input) {
            const input = document.getElementById('encoding-input');
            if (input) input.value = state.input;
        }

        if (state.method) {
```

```
        const select = document.getElementById('encoding-method');
        if (select) select.value = state.method;
    }

    if (state.result) {
        const result = document.getElementById('encoding-result');
        if (result) result.innerHTML = state.result;
    }
} catch (e) {
    console.error('Error loading encoding state:', e);
}
}

async function saveTransactionState() {
    const container = document.getElementById('transactions-operations');
    if (!container) return null;

    try {
        // Capture canvas state as image
        const canvas = document.getElementById('drawing-canvas');
        let canvasDataUrl = null;

        if (canvas) {
            canvasDataUrl = canvas.toDataURL('image/png');
        }

        return {
            canvasState: canvasDataUrl,
            history: document.getElementById('transaction-history')?.innerHTML
        };
    } catch (e) {
        console.error('Error saving transaction state:', e);
        return null;
    }
}

async function loadTransactionState(state) {
    if (!state) return;

    try {
        // Restore canvas state if possible
        if (state.canvasState) {
            const canvas = document.getElementById('drawing-canvas');
            if (canvas) {
                const ctx = canvas.getContext('2d');
                if (ctx) {
                    const img = new Image();
                    img.onload = function() {
                        ctx.clearRect(0, 0, canvas.width, canvas.height);
                        ctx.drawImage(img, 0, 0);
                    };
                    img.src = state.canvasState;
                }
            }
        }
    }
}
```

```
        }

        // Restore history display
        if (state.history) {
            const history = document.getElementById('transaction-history');
            if (history) history.innerHTML = state.history;
        }
    } catch (e) {
    console.error('Error loading transaction state:', e);
}
}

// ----- Programming Language -----


async function saveOperatorLangState() {
    const container = document.getElementById('programming-section');
    if (!container) return null;

    try {
        // Get editor content
        const editorContent = document.getElementById('code-editor')?.value;

        // Get console output
        const consoleOutput = Array.from(
            document.getElementById('language-console')?.childNodes || []
        ).map(node => node.textContent).join('\n');

        // Get canvas state
        const canvas = document.getElementById('language-canvas');
        let canvasDataUrl = null;

        if (canvas) {
            canvasDataUrl = canvas.toDataURL('image/png');
        }

        // Get variables from operatorLang if possible
        let variables = [];
        if (window.operatorLang && window.operatorLang.variables) {
            variables = Array.from(window.operatorLang.variables.entries());
        }

        return {
            code: editorContent,
            console: consoleOutput,
            canvasState: canvasDataUrl,
            variables: variables
        };
    } catch (e) {
    console.error('Error saving OperatorLang state:', e);
    return null;
}
}

async function loadOperatorLangState(state) {
```

```
if (!state) return;

try {
    // Restore editor content
    if (state.code) {
        const editor = document.getElementById('code-editor');
        if (editor) editor.value = state.code;
    }

    // Restore console output
    if (state.console) {
        const consoleEl = document.getElementById('language-console');
        if (consoleEl) {
            consoleEl.innerHTML = '';
            state.console.split('\n').forEach(line => {
                const lineElement = document.createElement('div');
                lineElement.className = 'console-line';
                lineElement.textContent = line;
                consoleEl.appendChild(lineElement);
            });
        }
    }

    // Restore canvas state
    if (state.canvasState) {
        const canvas = document.getElementById('language-canvas');
        if (canvas) {
            const ctx = canvas.getContext('2d');
            if (ctx) {
                const img = new Image();
                img.onload = function() {
                    ctx.clearRect(0, 0, canvas.width, canvas.height);
                    ctx.drawImage(img, 0, 0);
                };
                img.src = state.canvasState;
            }
        }
    }

    // Restore variables if possible
    if (state.variables && window.operatorLang) {
        try {
            window.operatorLang.variables = new Map(state.variables);
        } catch (e) {
            console.warn('Could not restore OperatorLang variables:', e);
        }
    }
} catch (e) {
    console.error('Error loading OperatorLang state:', e);
}

// Add this function to operator-persistence.js
function setupSessionManagementListeners() {
```

```
// Save workspace button
const saveWorkspaceBtn = document.getElementById('save-workspace-btn');
if (saveWorkspaceBtn) {
    saveWorkspaceBtn.addEventListener('click', saveAllStates);
}

// Load workspace button
const loadWorkspaceBtn = document.getElementById('load-workspace-btn');
if (loadWorkspaceBtn) {
    loadWorkspaceBtn.addEventListener('click', loadFromLocalStorage);
}

// Export workspace button
const exportWorkspaceBtn = document.getElementById('export-workspace-
btn');
if (exportWorkspaceBtn) {
    exportWorkspaceBtn.addEventListener('click', exportWorkspace);
}

// Import workspace button
const importWorkspaceBtn = document.getElementById('import-workspace-
btn');
if (importWorkspaceBtn) {
    importWorkspaceBtn.addEventListener('click', importWorkspace);
}

// Auto-save toggle
const autoSaveToggle = document.getElementById('auto-save-toggle');
const autoSaveLabel = document.getElementById('auto-save-label');
if (autoSaveToggle && autoSaveLabel) {
    // Set initial state
    const autoSaveEnabled = localStorage.getItem(CONFIG.autoSaveKey) ===
'true';
    autoSaveToggle.checked = autoSaveEnabled;
    autoSaveLabel.textContent = `Auto-save is ${autoSaveEnabled ? 'on' :
'off'}`;

    // Add change listener
    autoSaveToggle.addEventListener('change', function() {
        const newState = this.checked;
        localStorage.setItem(CONFIG.autoSaveKey, newState.toString());
        autoSaveLabel.textContent = `Auto-save is ${newState ? 'on' :
'off'}`;
        showNotification(`Auto-save ${newState ? 'enabled' : 'disabled'}`, 'info');
    });
}

// Clear storage button
const clearStorageBtn = document.getElementById('clear-storage-btn');
if (clearStorageBtn) {
    clearStorageBtn.addEventListener('click', function() {
        if (confirm('Are you sure you want to clear all saved workspace
data? This cannot be undone.')) {
```

```
    clearAllStoredData();
    updateWorkspaceInfo();
    showNotification('All stored workspace data has been cleared',
'info');
}
})
}
}

// Add this function to operator-persistence.js
async function collectAllStates() {
    const fullState = {
        version: CONFIG.version,
        timestamp: new Date().toISOString(),
        components: {}
    };

    // Save each component's state
    for (const [id, component] of stateRegistry.entries()) {
        if (!component.active) continue;

        try {
            const componentState = await component.save();
            if (componentState !== undefined) {
                fullState.components[id] = componentState;
            }
        } catch (error) {
            console.error(`Error saving state for ${id}:`, error);
        }
    }

    return fullState;
}

// Add these functions to operator-persistence.js
function exportWorkspace() {
    safeExecute(async () => {
        // Collect all state
        const fullState = await collectAllStates();

        // Create a JSON file
        const blob = new Blob([JSON.stringify(fullState, null, 2)], {type:
'application/json'});
        const url = URL.createObjectURL(blob);

        // Create download link
        const a = document.createElement('a');
        a.href = url;
        a.download = `operator_workspace_${new
Date().toISOString().replace(/:/g, '-')}.json`;
        a.click();

        // Clean up
        setTimeout(() => URL.revokeObjectURL(url), 5000);
    });
}
```

```
// Add to history
addHistoryItem('Workspace exported to file');

showNotification('Workspace exported successfully', 'success');
return 'Workspace exported successfully';
}, error => {
    showNotification('Failed to export workspace: ' + error.message,
'error');
});

function importWorkspace() {
    // Create file input
    const fileInput = document.createElement('input');
    fileInput.type = 'file';
    fileInput.accept = '.json';
    fileInput.style.display = 'none';
    document.body.appendChild(fileInput);

    fileInput.addEventListener('change', function(e) {
        const file = e.target.files[0];
        if (!file) return;

        const reader = new FileReader();
        reader.onload = function(e) {
            safeExecute(async () => {
                // Parse the file
                const data = JSON.parse(e.target.result);

                // Validate
                if (!data.timestamp) {
                    throw new Error('Invalid workspace file format');
                }

                // Confirm
                if (!confirm(`Load workspace from ${new
Date(data.timestamp).toLocaleString()}? This will replace your current
workspace.`)) {
                    return;
                }

                // Load the data
                await loadFromObject(data);

                // Add to history
                addHistoryItem('Workspace imported from file');

                showNotification('Workspace imported successfully',
'success');
                updateWorkspaceInfo();

                return 'Workspace imported successfully';
            }, error => {
```

```
        showNotification('Failed to import workspace: ' +
error.message, 'error');
    });
}

reader.readAsText(file);

// Clean up
document.body.removeChild(fileInput);
});

fileInput.click();
}

function clearAllStoredData() {
// Clear localStorage
const metadataKey = localStorage.getItem(CONFIG.metadataKey);
if (metadataKey) {
try {
    const metadata = JSON.parse(metadataKey);
    // Remove chunks
    for (let i = 0; i < metadata.chunks; i++) {
        localStorage.removeItem(`#${CONFIG.chunkPrefix}${i}`);
    }
} catch (e) {
    console.error('Error parsing metadata:', e);
}
}

// Remove metadata
localStorage.removeItem(CONFIG.metadataKey);
}

// Add to history
addHistoryItem('Cleared all stored workspace data');
}

function addHistoryItem(action) {
const historyContainer = document.getElementById('session-history');
if (!historyContainer) return;

// Remove empty history message if present
const emptyHistory = historyContainer.querySelector('.empty-history');
if (emptyHistory) {
    historyContainer.removeChild(emptyHistory);
}

// Create history item
const historyItem = document.createElement('div');
historyItem.className = 'history-item';

const now = new Date();
historyItem.innerHTML =
<div class="history-time">${now.toLocaleTimeString()}</div>
<div class="history-action">${action}</div>
```

```
`;

// Add to container (at the top)
if (historyContainer.firstChild) {
    historyContainer.insertBefore(historyItem,
historyContainer.firstChild);
} else {
    historyContainer.appendChild(historyItem);
}

// Limit history items
const maxItems = 10;
const items = historyContainer.querySelectorAll('.history-item');
if (items.length > maxItems) {
    for (let i = maxItems; i < items.length; i++) {
        historyContainer.removeChild(items[i]);
    }
}
}

function updateWorkspaceInfo() {
    const lastSavedTime = document.getElementById('last-saved-time');
    const componentCount = document.getElementById('component-count');
    const storageUsed = document.getElementById('storage-used');

    if (!lastSavedTime || !componentCount || !storageUsed) return;

    // Get last saved time
    const metadataStr = localStorage.getItem(CONFIG.metadataKey);
    if (metadataStr) {
        try {
            const metadata = JSON.parse(metadataStr);
            lastSavedTime.textContent = new
Date(metadata.timestamp).toLocaleString();
            componentCount.textContent = stateRegistry.size.toString();

            // Calculate storage used
            let size = metadata.size || 0;
            let sizeStr;

            if (size < 1024) {
                sizeStr = `${size} bytes`;
            } else if (size < 1024 * 1024) {
                sizeStr = ` ${(size / 1024).toFixed(2)} KB`;
            } else {
                sizeStr = ` ${(size / (1024 * 1024)).toFixed(2)} MB`;
            }

            storageUsed.textContent = sizeStr;
        } catch (e) {
            console.error('Error parsing metadata:', e);
            lastSavedTime.textContent = 'Error reading data';
        }
    } else {

```

```
lastSavedTime.textContent = 'Never';
componentCount.textContent = stateRegistry.size.toString();
storageUsed.textContent = '0 bytes';
    }
}

// Add this utility function
function safeExecute(func, errorCallback) {
    try {
        return func();
    } catch (error) {
        console.error('Operation error:', error);
        if (errorCallback) {
            errorCallback(error);
        }
        return null;
    }
}

// Add this to initPersistenceSystem function
function addSessionManagementStyles() {
    const styleElement = document.createElement('style');
    styleElement.textContent =
        /* Session Management Styles */
        '.workspace-info {
            background-color: var(--background-color);
            border: 1px solid var(--secondary-color);
            border-radius: 4px;
            padding: 15px;
        }

.info-item {
            display: flex;
            justify-content: space-between;
            margin-bottom: 10px;
            padding-bottom: 10px;
            border-bottom: 1px solid rgba(0,0,0,0.05);
        }

.info-item:last-child {
            margin-bottom: 0;
            padding-bottom: 0;
            border-bottom: none;
        }

.info-label {
            font-weight: bold;
            color: var(--primary-color);
        }

.session-history {
            max-height: 200px;
            overflow-y: auto;
            background-color: var(--background-color);
        }
    '
}
```

```
        border: 1px solid var(--secondary-color);
        border-radius: 4px;
        padding: 10px;
    }

.history-item {
    padding: 8px;
    border-bottom: 1px solid rgba(0,0,0,0.05);
}

.history-item:last-child {
    border-bottom: none;
}

.history-time {
    font-size: 0.8em;
    color: var(--secondary-color);
}

.history-action {
    font-weight: bold;
}

.empty-history {
    text-align: center;
    color: #888;
    padding: 20px 0;
}

.toggle-container {
    display: flex;
    align-items: center;
    gap: 10px;
    margin-bottom: 10px;
}

.toggle {
    position: relative;
    display: inline-block;
    width: 50px;
    height: 24px;
}

.toggle input {
    opacity: 0;
    width: 0;
    height: 0;
}

.toggle-slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
```

```
        right: 0;
        bottom: 0;
        background-color: #ccc;
        transition: .4s;
        border-radius: 24px;
    }

    .toggle-slider:before {
        position: absolute;
        content: "";
        height: 16px;
        width: 16px;
        left: 4px;
        bottom: 4px;
        background-color: white;
        transition: .4s;
        border-radius: 50%;
    }

    input:checked + .toggle-slider {
        background-color: var(--primary-color);
    }

    input:checked + .toggle-slider:before {
        transform: translateX(26px);
    }

    .hint-text {
        font-size: 0.8em;
        color: #666;
        margin-top: 5px;
    }

    .mt-20 {
        margin-top: 20px;
    }
};

document.head.appendChild(styleElement);

}

// Call this function
addSessionManagementStyles();

// Update the export object at the bottom of operator-persistence.js
window.OperatorPersistence = {
    saveAllStates,
    loadAllStates: loadFromLocalStorage,
    loadFromFile: loadWorkspaceFromFile,
    loadFromObject,
    exportWorkspace,
    importWorkspace,
    registerComponent,
```

```
    showNotification,
    updateWorkspaceInfo,
    addHistoryItem
};

})();
```

## [99] integerDatabase/operator.css

- **Bytes:** 14260
- **Type:** text

```
/* Base Styles and CSS Variables */
:root {
    --primary-color: #4B5320;          /* Army Green */
    --secondary-color: #BDB76B;         /* Dark Khaki */
    --background-color: #F5F5DC;        /* Beige */
    --text-color: #333333;             /* Dark Gray */
    --accent-color: #8B4513;           /* Saddle Brown */
    --success-color: #4CAF50;          /* Green */
    --error-color: #F44336;            /* Red */
    --warning-color: #FFC107;           /* Amber */
    --info-color: #2196F3;             /* Blue */
    --font-main: "Fira Code", "Consolas", monospace;
    --border-width: 2px;
    --shadow-standard: 0 2px 5px rgba(0, 0, 0, 0.2);
    --transition-standard: all 0.3s ease;
}

/* Global Reset */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

/* Body Styling */
body {
    font-family: var(--font-main);
    background-color: var(--background-color);
    color: var(--text-color);
    line-height: 1.6;
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    overflow-x: hidden;
}

/* Header Styling */
header {
    background-color: var(--primary-color);
```

```
color: white;
padding: 20px;
text-align: center;
box-shadow: var(--shadow-standard);
}

header h1 {
  font-size: 2.5rem;
  margin-bottom: 10px;
  text-transform: uppercase;
  letter-spacing: 1px;
}

header p {
  font-size: 1.1rem;
  opacity: 0.9;
}

/* Navigation and Main Layout */
nav#main-nav {
  background-color: var(--secondary-color);
  padding: 15px;
  display: flex;
  flex-wrap: wrap;
  justify-content: space-around;
  position: sticky;
  top: 0;
  z-index: 100;
  box-shadow: var(--shadow-standard);
}

.nav-section {
  padding: 8px;
  margin: 5px;
  min-width: 200px;
}

.nav-section h3 {
  color: var(--text-color);
  margin-bottom: 10px;
  font-size: 1.1rem;
  border-bottom: 1px solid var(--text-color);
  padding-bottom: 5px;
}

/* Buttons */
button {
  background-color: var(--primary-color);
  color: white;
  border: var(--border-width) solid var(--secondary-color);
  padding: 8px 15px;
  margin: 5px;
  cursor: pointer;
  font-family: var(--font-main);
}
```

```
    font-size: 0.9rem;
    transition: var(--transition-standard);
}

button:hover {
    background-color: var(--secondary-color);
    color: var(--text-color);
    transform: translateY(-2px);
}

button:active {
    transform: translateY(1px);
}

button:disabled {
    background-color: #cccccc;
    color: #666666;
    cursor: not-allowed;
    transform: none;
    border-color: #999999;
}

/* Main Content Area */
main {
    flex: 1;
    padding: 20px;
    max-width: 1200px;
    margin: 0 auto;
    width: 100%;
}

/* Operation Sections */
.operation-section {
    background-color: white;
    margin-bottom: 30px;
    border: var(--border-width) solid var(--secondary-color);
    box-shadow: var(--shadow-standard);
    display: none; /* Hidden by default, shown when selected */
}

.operation-section.active {
    display: block;
}

.operation-section h2 {
    background-color: var(--secondary-color);
    color: var(--text-color);
    padding: 15px;
    margin: 0;
    font-size: 1.5rem;
}

.operation-container {
    display: flex;
```

```
flex-wrap: wrap;
padding: 20px;
gap: 20px;
}

.operation-controls {
  flex: 1;
  min-width: 300px;
}

.operation-result {
  flex: 1;
  min-width: 300px;
}

.operation-controls h3,
.operation-result h3 {
  color: var(--primary-color);
  margin-bottom: 15px;
  padding-bottom: 5px;
  border-bottom: 1px solid var(--secondary-color);
}

/* Form Styling */
.form-group {
  margin-bottom: 15px;
}

.form-group label {
  display: block;
  margin-bottom: 5px;
  color: var(--text-color);
  font-weight: 500;
}

.form-group input,
.form-group select,
.form-group textarea {
  width: 100%;
  padding: 10px;
  border: 1px solid var(--secondary-color);
  font-family: var(--font-main);
  background-color: #fcfcf7;
  color: var(--text-color);
}

.form-group input:focus,
.form-group select:focus,
.form-group textarea:focus {
  outline: none;
  border-color: var(--primary-color);
  box-shadow: 0 0 5px rgba(75, 83, 32, 0.3);
}
```

```
.form-group textarea {
  min-height: 100px;
  resize: vertical;
}

.control-row {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  margin-top: 15px;
}

/* Result Containers */
.result-container {
  background-color: #fcfcf7;
  border: 1px solid var(--secondary-color);
  padding: 15px;
  min-height: 150px;
  max-height: 400px;
  overflow-y: auto;
  overflow-x: hidden;
  white-space: pre-wrap;
  word-break: break-word;
  font-family: var(--font-main);
}

/* Binary and Tree Visualizations */
.binary-display {
  display: flex;
  flex-direction: column;
  gap: 5px;
  font-family: var(--font-main);
  font-size: 0.9rem;
}

.binary-row {
  display: flex;
  align-items: center;
  gap: 5px;
}

.bit {
  width: 30px;
  height: 30px;
  display: flex;
  align-items: center;
  justify-content: center;
  border: 1px solid var(--secondary-color);
}

.bit.one {
  background-color: var(--primary-color);
  color: white;
}
```

```
.bit.zero {
  background-color: white;
  color: var(--text-color);
}

/* Quadtree Container */
#quadtree-container {
  position: relative;
  width: 100%;
  min-height: 400px;
  border: 2px solid var(--secondary-color);
  background-color: white;
  margin-top: 15px;
}

.quadtree-cell {
  position: absolute;
  border: 1px solid var(--secondary-color);
  background-color: var(--background-color);
  display: flex;
  justify-content: center;
  align-items: center;
  font-family: var(--font-main);
  color: var(--text-color);
  text-align: center;
  cursor: pointer;
  transition: transform 0.2s ease;
}

.quadtree-cell:hover {
  transform: scale(1.05);
  box-shadow: var(--shadow-standard);
}

.cell-content {
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  overflow: hidden;
}

/* Process Visualization */
.process {
  background-color: var(--background-color);
  border: 1px solid var(--secondary-color);
  padding: 10px;
  margin-bottom: 10px;
  position: relative;
}

.process-name {
```

```
    font-weight: bold;
    margin-bottom: 5px;
}

.process-progress {
    background-color: #e0e0e0;
    height: 10px;
    width: 100%;
    overflow: hidden;
}

.process-bar {
    background-color: var(--primary-color);
    height: 100%;
    width: 0%;
    transition: width 0.3s linear;
}

process-status {
    margin-top: 5px;
    font-size: 0.8rem;
}

process-close {
    position: absolute;
    top: 5px;
    right: 5px;
    cursor: pointer;
    color: var(--error-color);
    font-size: 1.2rem;
}

/* Database Visualization */
.database-record {
    background-color: var(--background-color);
    border: 1px solid var(--secondary-color);
    padding: 10px;
    margin-bottom: 10px;
}

.record-header {
    display: flex;
    justify-content: space-between;
    margin-bottom: 5px;
}

.record-id {
    font-weight: bold;
}

.record-controls {
    display: flex;
    gap: 5px;
}
```

```
.record-control {
  cursor: pointer;
  color: var(--primary-color);
  font-size: 0.8rem;
}

.record-content {
  padding: 5px;
  background-color: white;
}

/* File System Visualization */
#file-tree {
  list-style-type: none;
}

.file-item {
  padding: 5px;
  cursor: pointer;
}

.file-item:hover {
  background-color: var(--secondary-color);
}

.file-icon {
  margin-right: 5px;
}

.file-name {
  font-weight: bold;
}

.directory-item {
  font-weight: bold;
  cursor: pointer;
}

.directory-children {
  padding-left: 20px;
  list-style-type: none;
}

/* Memory Display */
.memory-file {
  margin-bottom: 15px;
}

.memory-file-title {
  font-weight: bold;
  padding-bottom: 5px;
  border-bottom: 1px solid var(--secondary-color);
}
```

```
.memory-slot {  
  padding: 8px;  
  border-bottom: 1px solid #f0f0e0;  
}  
  
.memory-slot:hover {  
  background-color: #f0f0e0;  
}  
  
.memory-slot-title {  
  font-weight: bold;  
  margin-bottom: 5px;  
}  
  
.memory-slot-value {  
  padding: 5px;  
  background-color: #f8f8f8;  
  border-radius: 2px;  
  font-size: 0.9em;  
}  
  
/* Progress Bar */  
.progress-bar {  
  background-color: #e0e0e0;  
  height: 20px;  
  width: 100%;  
  margin: 10px 0;  
  overflow: hidden;  
}  
  
.progress-fill {  
  background-color: var(--primary-color);  
  height: 100%;  
  width: 0%;  
  transition: width 0.3s linear;  
}  
  
/* Canvas Styling */  
#drawing-canvas {  
  border: 2px solid var(--secondary-color);  
  cursor: crosshair;  
  background-color: white;  
  width: 100%;  

```

```
}

.transaction-item {
  padding: 3px 5px;
  margin: 2px 0;
  background-color: #f0f0e0;
}

/* Modal Styling */
.modal {
  display: none;
  position: fixed;
  z-index: 1000;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.7);
  overflow: auto;
}

.modal-content {
  background-color: white;
  margin: 5% auto;
  padding: 20px;
  border: 1px solid var(--secondary-color);
  width: 80%;
  max-width: 600px;
  box-shadow: var(--shadow-standard);
  position: relative;
}

.close-btn {
  color: var(--secondary-color);
  position: absolute;
  top: 10px;
  right: 15px;
  font-size: 28px;
  font-weight: bold;
  cursor: pointer;
}

.close-btn:hover {
  color: var(--primary-color);
}

/* Status Messages */
.status-message {
  padding: 10px;
  margin: 10px 0;
  border-radius: 4px;
}

.status-success {
```

```
background-color: rgba(76, 175, 80, 0.2);
border: 1px solid var(--success-color);
color: var(--success-color);
}

.status-error {
background-color: rgba(244, 67, 54, 0.2);
border: 1px solid var(--error-color);
color: var(--error-color);
}

.status-warning {
background-color: rgba(255, 193, 7, 0.2);
border: 1px solid var(--warning-color);
color: var(--warning-color);
}

.status-info {
background-color: rgba(33, 150, 243, 0.2);
border: 1px solid var(--info-color);
color: var(--info-color);
}

/* Compression Stats Display */
#compression-stats {
margin-top: 10px;
padding: 10px;
background-color: #f0f0e0;
border: 1px solid var(--secondary-color);
}

/* Footer Styling */
footer {
background-color: var(--primary-color);
color: white;
text-align: center;
padding: 15px;
margin-top: 30px;
font-size: 0.9rem;
}

/* Responsive Design */
@media (max-width: 992px) {
.operation-container {
flex-direction: column;
}

.operation-controls,
.operation-result {
width: 100%;
}

.nav-section {
width: 100%;
```

```
}

.modal-content {
    width: 95%;
}

}

@media (max-width: 768px) {
    header h1 {
        font-size: 2rem;
    }

    .control-row {
        flex-direction: column;
    }

    .control-row button {
        width: 100%;
    }
}

@media (max-width: 480px) {
    body {
        font-size: 14px;
    }

    header h1 {
        font-size: 1.8rem;
    }

    .operation-section h2 {
        font-size: 1.2rem;
    }
}

/* Animations */
@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

.fade-in {
    animation: fadeIn 0.5s ease forwards;
}

@keyframes pulse {
    0% {
        transform: scale(1);
```

```
        }
      50% {
        transform: scale(1.05);
      }
      100% {
        transform: scale(1);
      }
    }

.pulse {
  animation: pulse 1s infinite;
}

/* Utility Classes */
.hidden {
  display: none !important;
}

.text-center {
  text-align: center;
}

.text-right {
  text-align: right;
}

.mt-10 {
  margin-top: 10px;
}

.mb-10 {
  margin-bottom: 10px;
}

.mt-20 {
  margin-top: 20px;
}

.mb-20 {
  margin-bottom: 20px;
}

.flex-row {
  display: flex;
}

.justify-between {
  justify-content: space-between;
}

.align-center {
  align-items: center;
}
```

```
/* Session persistence styling */
.persistence-controls {
  display: flex;
  flex-direction: column;
  gap: 5px;
}

.persistence-controls button {
  width: 100%;
  padding: 8px;
  font-size: 0.9rem;
}

.notification {
  position: fixed;
  top: 20px;
  right: 20px;
  z-index: 9999;
  background-color: white;
  border: 2px solid var(--secondary-color);
  border-left: 5px solid var(--primary-color);
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
  padding: 15px;
  max-width: 350px;
  font-size: 0.9rem;
  animation: slideIn 0.3s ease-out;
}

.success-notification {
  border-left: 5px solid var(--success-color);
}

.notification-content p {
  margin: 0 0 10px 0;
}

.notification-actions {
  display: flex;
  gap: 10px;
  justify-content: flex-end;
}

.notification-actions button {
  padding: 5px 10px;
  font-size: 0.8rem;
  background-color: var(--primary-color);
  color: white;
  border: none;
  cursor: pointer;
}

.notification-actions button:last-child {
  background-color: var(--secondary-color);
}
```

```
.fade-out {
    opacity: 0;
    transition: opacity 0.3s;
}

@keyframes slideIn {
    from {
        transform: translateX(100px);
        opacity: 0;
    }
    to {
        transform: translateX(0);
        opacity: 1;
    }
}
```

## [100] integerDatabase/operator.html

- **Bytes:** 29881
- **Type:** text

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Operator System - Integrated CS Principles</title>
    <link rel="stylesheet" href="operator.css">
    <!-- Math.js library for expression evaluation -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/11.8.0/math.min.js"></script>
    <!-- JSZip library for file compression -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jszip/3.10.1/jszip.min.js"></script>
</head>
<body>
    <header>
        <h1>Operator System</h1>
        <p>Integrated Computer Science, Operating System, and CRUD Database Principles</p>
    </header>

    <nav id="main-nav">
        <div class="nav-section">
            <h3>Data Structures</h3>
            <button data-section="array-operations">Arrays</button>
            <button data-section="object-operations">Objects</button>
            <button data-section="set-operations">Sets</button>
            <button data-section="tree-operations">Trees (Quadtree)</button>
        </div>
    </nav>

```

```
<div class="nav-section">
    <h3>Algorithms</h3>
    <button data-section="recursion-operations">Recursion</button>
    <button data-section="iteration-operations">Iteration</button>
    <button data-section="search-operations">Search</button>
    <button data-section="binary-operations">Binary Operations</button>
</div>
<div class="nav-section">
    <h3>System Operations</h3>
    <button data-section="file-operations">File System</button>
    <button data-section="memory-operations">Memory Management</button>
    <button data-section="process-operations">Process Management</button>
</div>
<div class="nav-section">
    <h3>CRUD Operations</h3>
    <button data-section="create-operations">Create</button>
    <button data-section="read-operations">Read</button>
    <button data-section="update-operations">Update</button>
    <button data-section="delete-operations">Delete</button>
</div>
<div class="nav-section">
    <h3>Advanced Concepts</h3>
    <button data-section="serialization-operations">Serialization</button>
    <button data-section="compression-operations">Compression</button>
    <button data-section="encoding-operations">Binary Encoding</button>
    <button data-section="transactions-operations">Transactions</button>
</div>
</nav>

<main id="main">
    <!-- Section for Array Operations -->
    <section id="array-operations" class="operation-section">
        <h2>Array Operations</h2>
        <div class="operation-container">
            <div class="operation-controls">
                <h3>Create Array</h3>
                <textarea id="array-input" placeholder="Enter comma-separated values (e.g. 1,2,3,4,5)"></textarea>
                <div class="control-row">
                    <button id="create-array">Create Array</button>
                    <button id="sort-array">Sort</button>
                    <button id="filter-array">Filter > 5</button>
                    <button id="map-array">Map (x2)</button>
                    <button id="reduce-array">Sum</button>
                </div>
            </div>
            <div class="operation-result">
                <h3>Result</h3>
                <div id="array-result" class="result-container"></div>
            </div>
        </div>
    </section>
    <!-- Section for Object Operations -->
```

```
<section id="object-operations" class="operation-section">
    <h2>Object Operations</h2>
    <div class="operation-container">
        <div class="operation-controls">
            <h3>Create Object</h3>
            <div class="form-group">
                <label for="object-key">Key:</label>
                <input type="text" id="object-key" placeholder="Enter
property name">
            </div>
            <div class="form-group">
                <label for="object-value">Value:</label>
                <input type="text" id="object-value" placeholder="Enter
property value">
            </div>
            <div class="control-row">
                <button id="add-property">Add Property</button>
                <button id="remove-property">Remove Property</button>
                <button id="clear-object">Clear Object</button>
            </div>
        </div>
        <div class="operation-result">
            <h3>Object Contents</h3>
            <div id="object-result" class="result-container"></div>
        </div>
    </div>
</section>

<!-- Section for Set Operations --&gt;
&lt;section id="set-operations" class="operation-section"&gt;
    &lt;h2&gt;Set Operations&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Create Sets&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="set-a-input"&gt;Set A:&lt;/label&gt;
                &lt;input type="text" id="set-a-input" placeholder="Enter
comma-separated values"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="set-b-input"&gt;Set B:&lt;/label&gt;
                &lt;input type="text" id="set-b-input" placeholder="Enter
comma-separated values"&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="set-union"&gt;Union&lt;/button&gt;
                &lt;button id="set-intersection"&gt;Intersection&lt;/button&gt;
                &lt;button id="set-difference"&gt;Difference (A-B)&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Result&lt;/h3&gt;
            &lt;div id="set-result" class="result-container"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;</pre>
```

```
</div>
</section>

<!-- Section for Tree (Quadtree) Operations --&gt;
&lt;section id="tree-operations" class="operation-section"&gt;
    &lt;h2&gt;Quadtree Operations&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Quadtree Settings&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="max-depth"&gt;Max Depth:&lt;/label&gt;
                &lt;input type="number" id="max-depth" min="1" max="6" value="3"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="quadtree-size"&gt;Size (px):&lt;/label&gt;
                &lt;input type="number" id="quadtree-size" min="200" max="800" value="400"&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="create-quadtree"&gt;Create Quadtree&lt;/button&gt;
                &lt;button id="reset-quadtree"&gt;Reset&lt;/button&gt;
                &lt;button id="export-quadtree"&gt;Export PNG&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;div id="quadtree-container"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;

<!-- Section for Recursion Operations --&gt;
&lt;section id="recursion-operations" class="operation-section"&gt;
    &lt;h2&gt;Recursion Operations&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Recursive Functions&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="recursion-n"&gt;Value (n):&lt;/label&gt;
                &lt;input type="number" id="recursion-n" min="1" max="40" value="5"&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="calc-factorial"&gt;Factorial&lt;/button&gt;
                &lt;button id="calc-fibonacci"&gt;Fibonacci&lt;/button&gt;
                &lt;button id="generate-tree"&gt;Generate Tree&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Result&lt;/h3&gt;
            &lt;div id="recursion-result" class="result-container"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;</pre>
```

```
<!-- Section for Iteration Operations -->
<section id="iteration-operations" class="operation-section">
    <h2>Iteration Operations</h2>
    <div class="operation-container">
        <div class="operation-controls">
            <h3>Generate Data</h3>
            <div class="form-group">
                <label for="iteration-n">Size (n):</label>
                <input type="number" id="iteration-n" min="1" max="10"
value="3">
            </div>
            <div class="form-group">
                <label for="charset-input">Character Set:</label>
                <input type="text" id="charset-input" value="abcdefghijklm">
            </div>
            <div class="control-row">
                <button id="generate-combinations">Generate
Combinations</button>
                <button id="stop-generation">Stop</button>
            </div>
        </div>
        <div class="operation-result">
            <h3>Generated Data</h3>
            <div id="iteration-progress" class="progress-bar">
                <div class="progress-fill"></div>
            </div>
            <div id="iteration-result" class="result-container">
                <p>Generated combinations will appear here</p>
            </div>
        </div>
    </div>
</section>

<!-- Section for Search Operations -->
<section id="search-operations" class="operation-section">
    <h2>Search Operations</h2>
    <div class="operation-container">
        <div class="operation-controls">
            <h3>Search Data</h3>
            <div class="form-group">
                <label for="search-data">Dataset:</label>
                <textarea id="search-data" placeholder="Enter comma-
separated values"></textarea>
            </div>
            <div class="form-group">
                <label for="search-query">Search For:</label>
                <input type="text" id="search-query" placeholder="Enter
search term">
            </div>
            <div class="control-row">
                <button id="linear-search">Linear Search</button>
                <button id="binary-search">Binary Search</button>
                <button id="fuzzy-search">Fuzzy Search</button>
            </div>
        </div>
    </div>
</section>
```

```
        </div>
    </div>
    <div class="operation-result">
        <h3>Search Results</h3>
        <div id="search-result" class="result-container"></div>
    </div>
</div>
</section>

<!-- Section for Binary Operations --&gt;
&lt;section id="binary-operations" class="operation-section"&gt;
    &lt;h2&gt;Binary Operations&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Binary Manipulation&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="binary-input-a"&gt;Value A:&lt;/label&gt;
                &lt;input type="number" id="binary-input-a" min="0" max="255"
value="42"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="binary-input-b"&gt;Value B:&lt;/label&gt;
                &lt;input type="number" id="binary-input-b" min="0" max="255"
value="27"&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="binary-and"&gt;AND&lt;/button&gt;
                &lt;button id="binary-or"&gt;OR&lt;/button&gt;
                &lt;button id="binary-xor"&gt;XOR&lt;/button&gt;
                &lt;button id="binary-not"&gt;NOT (A)&lt;/button&gt;
                &lt;button id="binary-shift"&gt;Shift Left&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Binary Result&lt;/h3&gt;
            &lt;div id="binary-result" class="result-container"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;

<!-- Section for File System Operations --&gt;
&lt;section id="file-operations" class="operation-section"&gt;
    &lt;h2&gt;File System Operations&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;File Management&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="file-content"&gt;File Content:&lt;/label&gt;
                &lt;textarea id="file-content" placeholder="Enter file
content to save"&gt;&lt;/textarea&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="file-name"&gt;File Name:&lt;/label&gt;
                &lt;input type="text" id="file-name" placeholder="Enter file
name"&gt;
            &lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;</pre>
```

```
        name" value="data.txt">
            </div>
            <div class="control-row">
                <button id="save-file">Save to File</button>
                <button id="load-file">Load File</button>
                <input type="file" id="file-upload" style="display:none">
            </div>
            <div class="control-row">
                <button id="create-directory">Create Directory</button>
                <button id="create-zip">Create ZIP</button>
            </div>
        </div>
        <div class="operation-result">
            <h3>File System</h3>
            <div id="file-system" class="result-container">
                <ul id="file-tree"></ul>
            </div>
        </div>
    </div>
</section>

<!-- Section for Memory Operations --&gt;
&lt;section id="memory-operations" class="operation-section"&gt;
    &lt;h2&gt;Memory Management&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Memory Slots&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="memory-file"&gt;File Number:&lt;/label&gt;
                &lt;input type="number" id="memory-file" min="1" value="1"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="memory-slot"&gt;Slot Number:&lt;/label&gt;
                &lt;input type="number" id="memory-slot" min="1" value="1"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="memory-content"&gt;Content:&lt;/label&gt;
                &lt;textarea id="memory-content" placeholder="Enter slot content"&gt;&lt;/textarea&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="assign-slot"&gt;Assign Slot&lt;/button&gt;
                &lt;button id="read-slot"&gt;Read Slot&lt;/button&gt;
                &lt;button id="clear-all-slots"&gt;Clear All&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Memory Contents&lt;/h3&gt;
            &lt;div id="memory-display" class="result-container"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;

<!-- Section for Process Operations --&gt;</pre>
```

```
<section id="process-operations" class="operation-section">
    <h2>Process Management</h2>
    <div class="operation-container">
        <div class="operation-controls">
            <h3>Processes</h3>
            <div class="form-group">
                <label for="process-name">Process Name:</label>
                <input type="text" id="process-name" placeholder="Enter process name">
            </div>
            <div class="form-group">
                <label for="process-duration">Duration (ms):</label>
                <input type="number" id="process-duration" min="100" max="10000" value="2000">
            </div>
            <div class="control-row">
                <button id="start-process">Start Process</button>
                <button id="stop-all-processes">Stop All</button>
            </div>
        </div>
        <div class="operation-result">
            <h3>Process Status</h3>
            <div id="process-list" class="result-container"></div>
        </div>
    </div>
</section>

<!-- Section for Create Operations --&gt;
&lt;section id="create-operations" class="operation-section"&gt;
    &lt;h2&gt;CRUD - Create&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Create Record&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="record-id"&gt;ID:&lt;/label&gt;
                &lt;input type="text" id="record-id" placeholder="Enter record ID"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="record-name"&gt;Name:&lt;/label&gt;
                &lt;input type="text" id="record-name" placeholder="Enter record name"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="record-data"&gt;Data:&lt;/label&gt;
                &lt;textarea id="record-data" placeholder="Enter record data"&gt;&lt;/textarea&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="create-record"&gt;Create Record&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Database Contents&lt;/h3&gt;</pre>
```

```
        <div id="database-display" class="result-container"></div>
    </div>
</div>
</section>


<section id="read-operations" class="operation-section">
    <h2>CRUD - Read</h2>
    <div class="operation-container">
        <div class="operation-controls">
            <h3>Read Record</h3>
            <div class="form-group">
                <label for="read-id">Record ID:</label>
                <input type="text" id="read-id" placeholder="Enter record
ID to read">
            </div>
            <div class="control-row">
                <button id="read-record">Read Record</button>
                <button id="read-all-records">Read All Records</button>
                <button id="query-records">Query Records</button>
            </div>
        </div>
        <div class="operation-result">
            <h3>Read Results</h3>
            <div id="read-result" class="result-container"></div>
        </div>
    </div>
</section>


<section id="update-operations" class="operation-section">
    <h2>CRUD - Update</h2>
    <div class="operation-container">
        <div class="operation-controls">
            <h3>Update Record</h3>
            <div class="form-group">
                <label for="update-id">Record ID:</label>
                <input type="text" id="update-id" placeholder="Enter
record ID to update">
            </div>
            <div class="form-group">
                <label for="update-field">Field:</label>
                <select id="update-field">
                    <option value="name">Name</option>
                    <option value="data">Data</option>
                </select>
            </div>
            <div class="form-group">
                <label for="update-value">New Value:</label>
                <textarea id="update-value" placeholder="Enter new value">
</textarea>
            </div>
            <div class="control-row">
                <button id="update-record">Update Record</button>
            </div>
        </div>
    </div>
</section>
```

```
        </div>
    </div>
    <div class="operation-result">
        <h3>Update Result</h3>
        <div id="update-result" class="result-container"></div>
    </div>
</div>
</section>

<!-- Section for Delete Operations --&gt;
&lt;section id="delete-operations" class="operation-section"&gt;
    &lt;h2&gt;CRUD - Delete&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Delete Record&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="delete-id"&gt;Record ID:&lt;/label&gt;
                &lt;input type="text" id="delete-id" placeholder="Enter record ID to delete"&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="delete-record"&gt;Delete Record&lt;/button&gt;
                &lt;button id="delete-all-records"&gt;Delete All Records&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Delete Result&lt;/h3&gt;
            &lt;div id="delete-result" class="result-container"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;

<!-- Section for Serialization Operations --&gt;
&lt;section id="serialization-operations" class="operation-section"&gt;
    &lt;h2&gt;Serialization &amp; Deserialization&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Serialize Data&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="serialize-input"&gt;Input Object:&lt;/label&gt;
                &lt;textarea id="serialize-input" placeholder="Enter JSON object"&gt;&lt;/textarea&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="serialization-format"&gt;Format:&lt;/label&gt;
                &lt;select id="serialization-format"&gt;
                    &lt;option value="json"&gt;JSON&lt;/option&gt;
                    &lt;option value="xml"&gt;XML&lt;/option&gt;
                    &lt;option value="csv"&gt;CSV&lt;/option&gt;
                &lt;/select&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="serialize-data"&gt;Serialize&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;</pre>
```

```
        <button id="deserialize-data">Deserialize</button>
    </div>
</div>
<div class="operation-result">
    <h3>Serialized Data</h3>
    <div id="serialization-result" class="result-container"></div>
</div>
</div>
</section>

<!-- Section for Compression Operations --&gt;
&lt;section id="compression-operations" class="operation-section"&gt;
    &lt;h2&gt;Data Compression&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Compress Data&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="compression-input"&gt;Input Text:&lt;/label&gt;
                &lt;textarea id="compression-input" placeholder="Enter text to compress"&gt;&lt;/textarea&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="compression-method"&gt;Method:&lt;/label&gt;
                &lt;select id="compression-method"&gt;
                    &lt;option value="rle"&gt;Run-Length Encoding&lt;/option&gt;
                    &lt;option value="huffman"&gt;Huffman Coding&lt;/option&gt;
                    &lt;option value="lz"&gt;LZ Compression&lt;/option&gt;
                &lt;/select&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="compress-data"&gt;Compress&lt;/button&gt;
                &lt;button id="decompress-data"&gt;Decompress&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Compression Result&lt;/h3&gt;
            &lt;div id="compression-result" class="result-container"&gt;&lt;/div&gt;
            &lt;div id="compression-stats"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;

<!-- Section for Binary Encoding Operations --&gt;
&lt;section id="encoding-operations" class="operation-section"&gt;
    &lt;h2&gt;Binary Encoding/Decoding&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Encode/Decode&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="encoding-input"&gt;Input:&lt;/label&gt;
                &lt;textarea id="encoding-input" placeholder="Enter text to encode"&gt;&lt;/textarea&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;</pre>
```

```
<label for="encoding-method">Method:</label>
<select id="encoding-method">
    <option value="base64">Base64</option>
    <option value="hex">Hexadecimal</option>
    <option value="binary">Binary</option>
    <option value="custom">Custom ID</option>
</select>
</div>
<div class="control-row">
    <button id="encode-data">Encode</button>
    <button id="decode-data">Decode</button>
</div>
</div>
<div class="operation-result">
    <h3>Encoding Result</h3>
    <div id="encoding-result" class="result-container"></div>
</div>
</div>
</section>

<!-- Section for Transaction Operations --&gt;
&lt;section id="transactions-operations" class="operation-section"&gt;
    &lt;h2&gt;Transactions &amp; History&lt;/h2&gt;
    &lt;div class="operation-container"&gt;
        &lt;div class="operation-controls"&gt;
            &lt;h3&gt;Canvas Operations&lt;/h3&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="drawing-color"&gt;Color:&lt;/label&gt;
                &lt;input type="color" id="drawing-color" value="#000000"&gt;
            &lt;/div&gt;
            &lt;div class="form-group"&gt;
                &lt;label for="drawing-size"&gt;Size:&lt;/label&gt;
                &lt;input type="range" id="drawing-size" min="1" max="50"
value="5"&gt;
            &lt;/div&gt;
            &lt;div class="control-row"&gt;
                &lt;button id="transaction-undo"&gt;Undo&lt;/button&gt;
                &lt;button id="transaction-redo"&gt;Redo&lt;/button&gt;
                &lt;button id="transaction-clear"&gt;Clear&lt;/button&gt;
            &lt;/div&gt;
        &lt;/div&gt;
        &lt;div class="operation-result"&gt;
            &lt;h3&gt;Drawing Canvas&lt;/h3&gt;
            &lt;canvas id="drawing-canvas" width="400" height="300"&gt;&lt;/canvas&gt;
            &lt;div id="transaction-history"&gt;&lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;/section&gt;
&lt;/main&gt;

&lt;footer&gt;
    &lt;p&gt;Operator System - Comprehensive Computer Science Principles
Implementation&lt;/p&gt;
    &lt;p&gt;&amp;copy; 2025 - An educational tool demonstrating software engineering&lt;/p&gt;
</pre>
```

```

concepts</p>
</footer>

<!-- Error modal for handling and displaying errors --&gt;
&lt;div id="error-modal" class="modal"&gt;
    &lt;div class="modal-content"&gt;
        &lt;span class="close-btn"&gt;&amp;times;;&lt;/span&gt;
        &lt;h2&gt;Error&lt;/h2&gt;
        &lt;p id="error-message"&gt;&lt;/p&gt;
    &lt;/div&gt;
&lt;/div&gt;

<!-- Result modal for displaying large results --&gt;
&lt;div id="result-modal" class="modal"&gt;
    &lt;div class="modal-content"&gt;
        &lt;span class="close-btn"&gt;&amp;times;;&lt;/span&gt;
        &lt;h2&gt;Result&lt;/h2&gt;
        &lt;div id="modal-result"&gt;&lt;/div&gt;
    &lt;/div&gt;
&lt;/div&gt;
&lt;script src="operator-framework.js"&gt;&lt;/script&gt;
&lt!-- Add this before operator.js in operator.html --&gt;
&lt;script src="operator-language.js"&gt;&lt;/script&gt;
&lt;script src="operator.js"&gt;&lt;/script&gt;
&lt!-- Add this right after operator.js and operator-language.js --&gt;
&lt;script src="operator-extensions.js"&gt;&lt;/script&gt;
&lt;script src="operator-dropdown.js"&gt;&lt;/script&gt;
&lt;script src="operator-map.js"&gt;&lt;/script&gt;
&lt;script src="operator-alphabet.js"&gt;&lt;/script&gt;
&lt;script src="operator-build.js"&gt;&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>

```

## [101] integerDatabase/operator.js

- **Bytes:** 153742
- **Type:** text

```

/**
 * operator.js
 * Operator System - Integrated Computer Science Principles
 *
 * This JavaScript file implements various computer science principles:
 * - Data Structures (Arrays, Objects, Sets, Trees)
 * - Algorithms (Recursion, Iteration, Search, Binary Operations)
 * - System Operations (File System, Memory Management, Process Management)
 * - CRUD Database Operations
 * - Advanced Concepts (Serialization, Compression, Encoding, Transactions)
 */

```

```
// Immediately Invoked Function Expression (IIFE) to avoid global scope pollution
```

```
(function() {
    'use strict';

    // ===== UTILITY FUNCTIONS =====

    /**
     * Utility function to show an error modal
     */
    function showError(message) {
        const modal = document.getElementById('error-modal');
        const messageElement = document.getElementById('error-message');
        messageElement.textContent = message;
        modal.style.display = 'block';
    }

    /**
     * Utility function to show a result in the modal
     */
    function showResultModal(content) {
        const modal = document.getElementById('result-modal');
        const resultElement = document.getElementById('modal-result');
        resultElement.innerHTML = '';

        if (typeof content === 'string') {
            resultElement.textContent = content;
        } else {
            resultElement.appendChild(content);
        }

        modal.style.display = 'block';
    }

    /**
     * Creates a status message element
     */
    function createStatusMessage(message, type) {
        const statusElement = document.createElement('div');
        statusElement.className = `status-message status-${type}`;
        statusElement.textContent = message;
        return statusElement;
    }

    /**
     * Global error handler with try-catch pattern
     */
    function safeExecute(func, errorCallback) {
        try {
            return func();
        } catch (error) {
            console.error('Operation error:', error);
            if (errorCallback) {
                errorCallback(error);
            } else {
                showError(error.message || 'An unexpected error occurred');
            }
        }
    }
})()
```

```
        }
        return null;
    }
}

// ===== SECTION NAVIGATION =====

/** 
 * Set up section navigation
 */
function setupNavigation() {
    // Get all section buttons
    const sectionButtons = document.querySelectorAll('#main-nav button');

    // Add click event to each button
    sectionButtons.forEach(button => {
        button.addEventListener('click', () => {
            const sectionId = button.getAttribute('data-section');
            showSection(sectionId);
        });
    });
}

// Show the first section by default
if (sectionButtons.length > 0) {
    const firstSectionId = sectionButtons[0].getAttribute('data-section');
    showSection(firstSectionId);
}

/** 
 * Show a specific section and hide all others
 */
function showSection(sectionId) {
    // Hide all sections
    const allSections = document.querySelectorAll('.operation-section');
    allSections.forEach(section => {
        section.classList.remove('active');
    });

    // Show the selected section
    const selectedSection = document.getElementById(sectionId);
    if (selectedSection) {
        selectedSection.classList.add('active');
        selectedSection.classList.add('fade-in');
        setTimeout(() => {
            selectedSection.classList.remove('fade-in');
        }, 500);
    }
}

// ===== 1. DATA STRUCTURES =====

// ===== 1.1 Array Operations =====
function setupArrayOperations() {
```

```
const arrayInput = document.getElementById('array-input');
const createArrayBtn = document.getElementById('create-array');
const sortArrayBtn = document.getElementById('sort-array');
const filterArrayBtn = document.getElementById('filter-array');
const mapArrayBtn = document.getElementById('map-array');
const reduceArrayBtn = document.getElementById('reduce-array');
const resultContainer = document.getElementById('array-result');

// Current array state
let currentArray = [];

// Create array from input
createArrayBtn.addEventListener('click', () => {
    safeExecute() => {
        const input = arrayInput.value;
        currentArray = input.split(',').map(item => {
            const trimmed = item.trim();
            // Try to convert to number if possible
            const num = Number(trimmed);
            return isNaN(num) ? trimmed : num;
        });

        displayArray(currentArray);
        return `Created array with ${currentArray.length} elements`;
    }, error => {
        resultContainer.textContent = `Error creating array:
${error.message}`;
    });
});

// Sort the array
sortArrayBtn.addEventListener('click', () => {
    safeExecute() => {
        if (currentArray.length === 0) {
            throw new Error('Array is empty. Create an array first.');
        }

        // Check if array has all numbers or all strings for proper
        sorting
        const allNumbers = currentArray.every(item => typeof item ===
        'number');
        const allStrings = currentArray.every(item => typeof item ===
        'string');

        if (allNumbers) {
            currentArray.sort((a, b) => a - b);
        } else if (allStrings) {
            currentArray.sort();
        } else {
            // Mixed type sorting - convert all to strings
            currentArray.sort((a, b) =>
String(a).localeCompare(String(b)));
        }
    }
});
```

```
        displayArray(currentArray);
        return 'Array sorted successfully';
    }, error => {
        resultContainer.textContent = `Error sorting array:
${error.message}`;
    });
});

// Filter array elements > 5
filterArrayBtn.addEventListener('click', () => {
    safeExecute(() => {
        if (currentArray.length === 0) {
            throw new Error('Array is empty. Create an array first.');
        }

        const filtered = currentArray.filter(item => {
            // Only filter numerical values
            return typeof item === 'number' && item > 5;
        });

        displayArray(filtered);
        return `Filtered array to ${filtered.length} elements > 5`;
    }, error => {
        resultContainer.textContent = `Error filtering array:
${error.message}`;
    });
});

// Map array (multiply by 2)
mapArrayBtn.addEventListener('click', () => {
    safeExecute(() => {
        if (currentArray.length === 0) {
            throw new Error('Array is empty. Create an array first.');
        }

        const mapped = currentArray.map(item => {
            // Only multiply numbers, leave strings as is
            return typeof item === 'number' ? item * 2 : item;
        });

        displayArray(mapped);
        return 'Mapped array (numbers multiplied by 2)';
    }, error => {
        resultContainer.textContent = `Error mapping array:
${error.message}`;
    });
});

// Reduce array (sum)
reduceArrayBtn.addEventListener('click', () => {
    safeExecute(() => {
        if (currentArray.length === 0) {
            throw new Error('Array is empty. Create an array first.');
        }
    });
});
```

```
// Extract numbers from the array
const numbers = currentArray.filter(item => typeof item ===
'number');

if (numbers.length === 0) {
    throw new Error('No numerical values found in the array');
}

const sum = numbers.reduce((acc, curr) => acc + curr, 0);

resultContainer.innerHTML = '';
const resultElement = document.createElement('div');
resultElement.textContent = `Sum of numerical values: ${sum}`;
resultContainer.appendChild(resultElement);

return `Reduced array to sum: ${sum}`;
}, error => {
    resultContainer.textContent = `Error reducing array:
${error.message}`;
});
});

// Helper function to display the array
function displayArray(array) {
    resultContainer.innerHTML = '';

    const arrayElement = document.createElement('div');
    arrayElement.className = 'array-display';

    array.forEach((item, index) => {
        const itemElement = document.createElement('div');
        itemElement.className = 'array-item';
        itemElement.innerHTML = `[${index}]</span>: <span class="array-value">${item}</span>`;
        arrayElement.appendChild(itemElement);
    });

    resultContainer.appendChild(arrayElement);
}

// ===== 1.2 Object Operations =====
function setupObjectOperations() {
    const objectKey = document.getElementById('object-key');
    const objectValue = document.getElementById('object-value');
    const addPropertyBtn = document.getElementById('add-property');
    const removePropertyBtn = document.getElementById('remove-property');
    const clearObjectBtn = document.getElementById('clear-object');
    const resultContainer = document.getElementById('object-result');

    // Current object state
    let currentObject = {};
```

```
// Add a property to the object
addPropertyBtn.addEventListener('click', () => {
    safeExecute(() => {
        const key = objectKey.value.trim();
        const value = objectValue.value.trim();

        if (!key) {
            throw new Error('Property key cannot be empty');
        }

        // Try to parse the value if it looks like a number or boolean
        let parsedValue = value;

        if (value.toLowerCase() === 'true') {
            parsedValue = true;
        } else if (value.toLowerCase() === 'false') {
            parsedValue = false;
        } else if (!isNaN(Number(value))) {
            parsedValue = Number(value);
        }

        // Add or update the property
        currentObject[key] = parsedValue;

        // Clear inputs
        objectKey.value = '';
        objectValue.value = '';

        // Display updated object
        displayObject(currentObject);
        return `Property '${key}' added to object`;
    }, error => {
        resultContainer.textContent = `Error adding property:
${error.message}`;
    });
});

// Remove a property from the object
removePropertyBtn.addEventListener('click', () => {
    safeExecute(() => {
        const key = objectKey.value.trim();

        if (!key) {
            throw new Error('Property key cannot be empty');
        }

        if (!(key in currentObject)) {
            throw new Error(`Property '${key}' does not exist in the
object`);
        }

        // Remove the property
        delete currentObject[key];
    });
});
```

```
// Clear input
objectKey.value = '';

// Display updated object
displayObject(currentObject);
return `Property '${key}' removed from object`;
}, error => {
    resultContainer.textContent = `Error removing property:
${error.message}`;
});
});

// Clear the object
clearObjectBtn.addEventListener('click', () => {
    currentObject = {};
    displayObject(currentObject);
});

// Helper function to display the object
function displayObject(obj) {
    resultContainer.innerHTML = '';

    if (Object.keys(obj).length === 0) {
        resultContainer.textContent = 'Empty object: {}';
        return;
    }

    const objectElement = document.createElement('div');
    objectElement.className = 'object-display';

    // Create JSON representation with syntax highlighting
    const jsonPre = document.createElement('pre');
    jsonPre.className = 'json-display';

    // Format JSON with indentation
    const formattedJson = JSON.stringify(obj, null, 2);
    jsonPre.textContent = formattedJson;

    objectElement.appendChild(jsonPre);
    resultContainer.appendChild(objectElement);
}

// Initialize with empty object
displayObject(currentObject);
}

// ===== 1.3 Set Operations =====
function setupSetOperations() {
    const setAInput = document.getElementById('set-a-input');
    const setBInput = document.getElementById('set-b-input');
    const unionBtn = document.getElementById('set-union');
    const intersectionBtn = document.getElementById('set-intersection');
    const differenceBtn = document.getElementById('set-difference');
    const resultContainer = document.getElementById('set-result');
```

```
// Helper function to parse input into a Set
function parseSetInput(input) {
    const values = input.split(',').map(item => item.trim());
    return new Set(values);
}

// Helper function to display a Set
function displaySet(set, label) {
    const setElement = document.createElement('div');
    setElement.className = 'set-display';

    const setLabel = document.createElement('div');
    setLabel.className = 'set-label';
    setLabel.textContent = label;

    const setContent = document.createElement('div');
    setContent.className = 'set-content';
    setContent.textContent = Array.from(set).join(', ');

    setElement.appendChild(setLabel);
    setElement.appendChild(setContent);

    return setElement;
}

// Union operation
unionBtn.addEventListener('click', () => {
    safeExecute(() => {
        const setA = parseSetInput(setAInput.value);
        const setB = parseSetInput(setBInput.value);

        // Perform union operation
        const union = new Set([...setA, ...setB]);

        // Display the result
        resultContainer.innerHTML = '';
        resultContainer.appendChild(displaySet(setA, 'Set A:'));
        resultContainer.appendChild(displaySet(setB, 'Set B:'));
        resultContainer.appendChild(displaySet(union, 'Union (A ∪ B):'));

        return `Union operation complete: ${union.size} elements`;
    }, error => {
        resultContainer.textContent = `Error performing union:
${error.message}`;
    });
});

// Intersection operation
intersectionBtn.addEventListener('click', () => {
    safeExecute(() => {
        const setA = parseSetInput(setAInput.value);
        const setB = parseSetInput(setBInput.value);
```

```
// Perform intersection operation
const intersection = new Set(
    [...setA].filter(item => setB.has(item))
);

// Display the result
resultContainer.innerHTML = '';
resultContainer.appendChild(displaySet(setA, 'Set A:'));
resultContainer.appendChild(displaySet(setB, 'Set B:'));
resultContainer.appendChild(displaySet(intersection, 'Intersection
(A ∩ B):'));

return `Intersection operation complete: ${intersection.size}
elements`;
}, error => {
    resultContainer.textContent = `Error performing intersection:
${error.message}`;
});
});

// Difference operation
differenceBtn.addEventListener('click', () => {
    safeExecute(() => {
        const setA = parseSetInput(setAInput.value);
        const setB = parseSetInput(setBInput.value);

        // Perform difference operation (A - B)
        const difference = new Set(
            [...setA].filter(item => !setB.has(item))
        );

        // Display the result
        resultContainer.innerHTML = '';
        resultContainer.appendChild(displaySet(setA, 'Set A:'));
        resultContainer.appendChild(displaySet(setB, 'Set B:'));
        resultContainer.appendChild(displaySet(difference, 'Difference (A
- B):'));

        return `Difference operation complete: ${difference.size}
elements`;
}, error => {
    resultContainer.textContent = `Error performing difference:
${error.message}`;
});
});
});

// ===== 1.4 Tree (Quadtree) Operations =====
function setupTreeOperations() {
    const maxDepthInput = document.getElementById('max-depth');
    const quadtreeSizeInput = document.getElementById('quadtree-size');
    const createQuadtreeBtn = document.getElementById('create-quadtree');
    const resetQuadtreeBtn = document.getElementById('reset-quadtree');
    const exportQuadtreeBtn = document.getElementById('export-quadtree');
```

```
const quadtreeContainer = document.getElementById('quadtree-container');

// Quadtree state
let quadtreeRoot = null;
let maxDepth = 3;
let quadtreeSize = 400;

// QuadtreeNode class
class QuadtreeNode {
    constructor(x, y, size, depth) {
        this.x = x;
        this.y = y;
        this.size = size;
        this.depth = depth;
        this.children = [];
        this.data = {};
        this.element = this.createElement();
        this.updateVisualState();
    }

    createElement() {
        const cell = document.createElement('div');
        cell.className = 'quadtree-cell';

        cell.style.left = `${this.x}px`;
        cell.style.top = `${this.y}px`;
        cell.style.width = `${this.size}px`;
        cell.style.height = `${this.size}px`;

        const content = document.createElement('div');
        content.className = 'cell-content';
        content.textContent = `d: ${this.depth}`;
        cell.appendChild(content);

        cell.addEventListener('click', () => {
            this.handleClick();
        });
    }

    quadtreeContainer.appendChild(cell);
    return cell;
}

updateVisualState() {
    if (this.depth < maxDepth) {
        this.element.classList.add('can-subdivide');
        this.element.title = 'Click to subdivide';
    } else {
        this.element.classList.remove('can-subdivide');
        this.element.title = 'Max depth reached';
    }
}

// Update color based on depth
const hue = (this.depth * 30) % 360;
this.element.style.backgroundColor = `hsl(${hue}, 70%, 90%)`;
```

```
        this.element.style.borderColor = `hsl(${hue}, 70%, 60%)`;
    }

    handleClick() {
        if (this.children.length === 0 && this.depth < maxDepth) {
            this.subdivide();
        }
    }

    subdivide() {
        if (this.children.length > 0 || this.depth >= maxDepth) return
false;

        const childSize = this.size / 2;

        this.children.push(
            new QuadtreeNode(this.x, this.y, childSize, this.depth + 1),
            new QuadtreeNode(this.x + childSize, this.y, childSize,
this.depth + 1),
            new QuadtreeNode(this.x, this.y + childSize, childSize,
this.depth + 1),
            new QuadtreeNode(this.x + childSize, this.y + childSize,
childSize, this.depth + 1)
        );
    }

    this.element.style.display = 'none';
    return true;
}

// Create the initial quadtree
function initQuadtree() {
    // Clear the container
    quadtreeContainer.innerHTML = '';

    // Get current settings
    maxDepth = parseInt(maxDepthInput.value) || 3;
    quadtreeSize = parseInt(quadtreeSizeInput.value) || 400;

    // Set container size
    quadtreeContainer.style.width = `${quadtreeSize}px`;
    quadtreeContainer.style.height = `${quadtreeSize}px`;

    // Create root node
    quadtreeRoot = new QuadtreeNode(0, 0, quadtreeSize, 0);
}

// Export the quadtree as PNG
function exportQuadtreeAsPNG() {
    const canvas = document.createElement('canvas');
    const context = canvas.getContext('2d');

    canvas.width = quadtreeSize;
    canvas.height = quadtreeSize;
```

```
// Fill background
context.fillStyle = 'white';
context.fillRect(0, 0, canvas.width, canvas.height);

// Recursive function to draw nodes
function drawNode(node) {
    if (!node) return;

    // Only draw visible nodes
    if (node.element.style.display !== 'none') {
        // Get computed styles
        const styles = window.getComputedStyle(node.element);

        // Draw cell background
        context.fillStyle = styles.backgroundColor;
        context.fillRect(node.x, node.y, node.size, node.size);

        // Draw cell border
        context.strokeStyle = styles.borderColor;
        context.lineWidth = 1;
        context.strokeRect(node.x, node.y, node.size, node.size);

        // Draw text
        context.fillStyle = '#333';
        context.font = '12px sans-serif';
        context.textAlign = 'center';
        context.textBaseline = 'middle';
        context.fillText(`d: ${node.depth}`,
                        node.x + node.size/2,
                        node.y + node.size/2);
    }

    // Draw children
    for (const child of node.children) {
        drawNode(child);
    }
}

// Draw the quadtree
drawNode(quadtreeRoot);

// Create download link
const link = document.createElement('a');
link.download = 'quadtree.png';
link.href = canvas.toDataURL('image/png');
link.click();
}

// Event listeners
createQuadtreeBtn.addEventListener('click', initQuadtree);
resetQuadtreeBtn.addEventListener('click', initQuadtree);
exportQuadtreeBtn.addEventListener('click', exportQuadtreeAsPNG);
```

```
// Initialize
initQuadtree();
}

// ===== 2. ALGORITHMS =====

// ===== 2.1 Recursion Operations =====
function setupRecursionOperations() {
    const recursionInput = document.getElementById('recursion-n');
    const factorialBtn = document.getElementById('calc-factorial');
    const fibonacciBtn = document.getElementById('calc-fibonacci');
    const generateTreeBtn = document.getElementById('generate-tree');
    const resultContainer = document.getElementById('recursion-result');

    // Recursive factorial function
    function factorial(n) {
        // Base case
        if (n <= 1) return 1;

        // Recursive case
        return n * factorial(n - 1);
    }

    // Recursive Fibonacci function
    function fibonacci(n) {
        // Base cases
        if (n <= 0) return 0;
        if (n === 1) return 1;

        // Recursive case
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    // Generate recursive tree visualization
    function generateTree(n) {
        const container = document.createElement('div');
        container.className = 'tree-container';

        function createTreeNode(value, depth) {
            const node = document.createElement('div');
            node.className = 'tree-node';

            const nodeContent = document.createElement('div');
            nodeContent.className = 'tree-node-content';
            nodeContent.textContent = value;

            node.appendChild(nodeContent);

            // Add children if depth > 0
            if (depth > 0) {
                const children = document.createElement('div');
                children.className = 'tree-children';

                // Left child is depth-1
                children.appendChild(createTreeNode(value, depth - 1));
                node.appendChild(children);
            }
        }

        createTreeNode(n, n);
        container.appendChild(container);
    }
}
```

```
        children.appendChild(createTreeNode(depth-1, depth-1));

        // Right child is depth/2 (integer division)
        if (depth > 1) {
            children.appendChild(createTreeNode(Math.floor(depth/2),
Math.floor(depth/2)));
        }

        node.appendChild(children);
    }

    return node;
}

container.appendChild(createTreeNode(n, n));
return container;
}

// Calculate factorial button
factorialBtn.addEventListener('click', () => {
    safeExecute(() => {
        const n = parseInt(recursionInput.value);

        if (isNaN(n) || n < 0) {
            throw new Error('Please enter a non-negative integer');
        }

        if (n > 20) {
            throw new Error('Value too large (max 20)');
        }

        const result = factorial(n);

        resultContainer.innerHTML = '';
        const resultElement = document.createElement('div');
        resultElement.innerHTML = `<strong>Factorial of ${n}:</strong>
${result}`;

        // Add explanation
        const explanation = document.createElement('div');
        explanation.className = 'recursion-explanation';
        explanation.innerHTML = `<p>Factorial is calculated recursively
as:</p>
<p>factorial(${n}) = ${n} * factorial(${n-
1})</p>
<p>factorial(1) = 1 (base case)</p>`;

        resultContainer.appendChild(resultElement);
        resultContainer.appendChild(explanation);

        return `Calculated factorial of ${n}: ${result}`;
    }, error => {
        resultContainer.textContent = `Error calculating factorial:
${error.message}`;
    }
});
```

```
        });
    });

    // Calculate Fibonacci button
    fibonacciBtn.addEventListener('click', () => {
        safeExecute(() => {
            const n = parseInt(recursionInput.value);

            if (isNaN(n) || n < 0) {
                throw new Error('Please enter a non-negative integer');
            }

            if (n > 40) {
                throw new Error('Value too large (max 40). Recursive Fibonacci is inefficient for large values.');
            }

            const result = fibonacci(n);

            resultContainer.innerHTML = '';
            const resultElement = document.createElement('div');
            resultElement.innerHTML = `<strong>Fibonacci(${n}):</strong>
${result}`;

            // Add explanation
            const explanation = document.createElement('div');
            explanation.className = 'recursion-explanation';
            explanation.innerHTML = `<p>Fibonacci is calculated recursively as:</p>
<p>fibonacci(${n}) = fibonacci(${n-1}) + fibonacci(${n-2})</p>
<p>fibonacci(0) = 0, fibonacci(1) = 1 (base cases)</p>`;

            resultContainer.appendChild(resultElement);
            resultContainer.appendChild(explanation);

            return `Calculated Fibonacci(${n}): ${result}`;
        }, error => {
            resultContainer.textContent = `Error calculating Fibonacci: ${error.message}`;
        });
    });

    // Generate Tree button
    generateTreeBtn.addEventListener('click', () => {
        safeExecute(() => {
            const n = parseInt(recursionInput.value);

            if (isNaN(n) || n < 0) {
                throw new Error('Please enter a non-negative integer');
            }

            if (n > 5) {
```

```
        throw new Error('Value too large (max 5). Larger values create
very large trees.');
    }

    const treeVisualization = generateTree(n);

    resultContainer.innerHTML = '';
    resultContainer.appendChild(treeVisualization);

    return `Generated recursive tree with depth ${n}`;
}, error => {
    resultContainer.textContent = `Error generating tree:
${error.message}`;
});
});

}

// ===== 2.2 Iteration Operations =====
function setupIterationOperations() {
    const iterationInput = document.getElementById('iteration-n');
    const charsetInput = document.getElementById('charset-input');
    const generateBtn = document.getElementById('generate-combinations');
    const stopBtn = document.getElementById('stop-generation');
    const resultContainer = document.getElementById('iteration-result');
    const progressBar = document.getElementById('iteration-progress');
    const progressFill = progressBar.querySelector('.progress-fill');

    // Flag to control generation
    let isGenerating = false;

    // Generate all possible combinations of length n from charset
    function generateCombinations(charset, n) {
        const combinations = [];
        const charsetArray = Array.from(charset);

        // Total number of combinations
        const total = Math.pow(charsetArray.length, n);

        // Iterator function using batch processing for UI responsiveness
        async function* combinationIterator() {
            const batchSize = 1000;
            let count = 0;

            // For each possible combination
            for (let i = 0; i < total && isGenerating; i++) {
                let combination = '';
                let index = i;

                // Generate combination by extracting characters
                for (let j = 0; j < n; j++) {
                    combination = charsetArray[index % charsetArray.length] +
combination;
                    index = Math.floor(index / charsetArray.length);
                }
            }
        }
    }
}
```

```
        combinations.push(combination);
        count++;

        // Yield after each batch to allow UI updates
        if (count % batchSize === 0 || i === total - 1) {
            yield {
                progress: (i + 1) / total * 100,
                combinations: combinations.slice() // Copy current
state
            };
            await new Promise(resolve => setTimeout(resolve, 0));
        }
    }

    return {
        iterator: combinationIterator(),
        total
    };
}

// Generate button click handler
generateBtn.addEventListener('click', async () => {
    safeExecute(async () => {
        const n = parseInt(iterationInput.value);
        const charset = charsetInput.value;

        if (isNaN(n) || n < 1) {
            throw new Error('Please enter a positive integer for n');
        }

        if (charset.length === 0) {
            throw new Error('Charset cannot be empty');
        }

        if (n > 5 && charset.length > 5) {
            throw new Error('Combination too large. Try a smaller n or
charset.');
        }

        // Calculate total combinations
        const totalCombinations = Math.pow(charset.length, n);

        // Update UI
        resultContainer.innerHTML = `<p>Generating ${totalCombinations}
combinations...</p>`;
        progressFill.style.width = '0%';
        isGenerating = true;
        generateBtn.disabled = true;
        stopBtn.disabled = false;

        // Generate combinations
        const generator = generateCombinations(charset, n);
    });
});
```

```
// Process each batch
let result;
try {
    for await (result of generator.iterator) {
        if (!isGenerating) break;

        // Update progress bar
        progressFill.style.width = `${result.progress}%`;

        // Display current state (limited to first 100 for
        performance)
        const displayCombinations = result.combinations.slice(0,
        100);
        resultContainer.innerHTML = `<p>Generated
        ${result.combinations.length} of ${totalCombinations} combinations</p>
        <div class="combinations-
        list">${displayCombinations.join(', ')}${result.combinations.length > 100 ? '...' :
        ''}</div>`;
    }
} catch (e) {
    console.error('Generation error:', e);
    throw e;
}

// Update UI after completion
isGenerating = false;
generateBtn.disabled = false;
stopBtn.disabled = true;

if (result && result.combinations) {
    return `Generated ${result.combinations.length} combinations
    with n=${n} and charset="${charset}"`;
} else {
    return 'Generation stopped';
}
}, error => {
    isGenerating = false;
    generateBtn.disabled = false;
    stopBtn.disabled = true;
    resultContainer.innerHTML = `<p class="error">Error:
    ${error.message}</p>`;
});
});

// Stop button click handler
stopBtn.addEventListener('click', () => {
    isGenerating = false;
    generateBtn.disabled = false;
    stopBtn.disabled = true;
    resultContainer.innerHTML += '<p>Generation stopped by user</p>';
});
}
```

```
// ===== 2.3 Search Operations =====
function setupSearchOperations() {
    const searchData = document.getElementById('search-data');
    const searchQuery = document.getElementById('search-query');
    const linearSearchBtn = document.getElementById('linear-search');
    const binarySearchBtn = document.getElementById('binary-search');
    const fuzzySearchBtn = document.getElementById('fuzzy-search');
    const resultContainer = document.getElementById('search-result');

    // Parse the data input into an array
    function parseData() {
        return searchData.value.split(',').map(item => item.trim());
    }

    // Linear search implementation
    function linearSearch(arr, query) {
        const results = [];
        const startTime = performance.now();

        // Search for the query
        for (let i = 0; i < arr.length; i++) {
            // Record number of comparisons
            let comparisons = 1;
            if (arr[i] === query) {
                results.push({
                    index: i,
                    value: arr[i],
                    comparisons
                });
            }
        }

        const endTime = performance.now();

        return {
            found: results.length > 0,
            results,
            time: endTime - startTime
        };
    }

    // Binary search implementation (requires sorted array)
    function binarySearch(arr, query) {
        // Sort array if it contains all numbers or all strings
        const isNumber = arr.every(item => !isNaN(Number(item)));
        const sorted = [...arr];

        // Sort the array
        if (isNumber) {
            sorted.sort((a, b) => Number(a) - Number(b));
        } else {
            sorted.sort();
        }
    }
}
```

```
const startTime = performance.now();
let comparisons = 0;

// Binary search algorithm
let left = 0;
let right = sorted.length - 1;
let foundIndex = -1;

while (left <= right) {
    const mid = Math.floor((left + right) / 2);
    comparisons++;

    if (sorted[mid] === query) {
        foundIndex = mid;
        break;
    } else if (sorted[mid] < query) {
        left = mid + 1;
    } else {
        right = mid - 1;
    }
}

const endTime = performance.now();

// Map back to original array if found
let results = [];
if (foundIndex !== -1) {
    const originalIndex = arr.indexOf(sorted[foundIndex]);
    results.push({
        index: originalIndex,
        value: arr[originalIndex],
        comparisons
    });
}

return {
    found: foundIndex !== -1,
    results,
    time: endTime - startTime,
    sortedArray: sorted
};

}

// Fuzzy search implementation
function fuzzySearch(arr, query) {
    const results = [];
    const startTime = performance.now();

    // Search for partial matches
    for (let i = 0; i < arr.length; i++) {
        let comparisons = 1;
        const strValue = String(arr[i]);
        const strQuery = String(query);
```

```
// Check if item contains the query
if (strValue.includes(strQuery)) {
    results.push({
        index: i,
        value: arr[i],
        comparisons,
        matchType: 'contains'
    });
}

// Check if item is "close" to query using Levenshtein distance
else {
    const distance = levenshteinDistance(strValue, strQuery);
    if (distance <= 2) { // Consider matches with distance <= 2
        results.push({
            index: i,
            value: arr[i],
            comparisons: comparisons + 1,
            matchType: 'similar',
            distance
        });
    }
}

const endTime = performance.now();

return {
    found: results.length > 0,
    results,
    time: endTime - startTime
};
}

// Levenshtein distance for fuzzy matching
function levenshteinDistance(a, b) {
    const matrix = [];

    // Initialize the matrix
    for (let i = 0; i <= b.length; i++) {
        matrix[i] = [i];
    }

    for (let j = 0; j <= a.length; j++) {
        matrix[0][j] = j;
    }

    // Fill the matrix
    for (let i = 1; i <= b.length; i++) {
        for (let j = 1; j <= a.length; j++) {
            if (b.charAt(i - 1) === a.charAt(j - 1)) {
                matrix[i][j] = matrix[i - 1][j - 1];
            } else {
                matrix[i][j] = Math.min(
                    matrix[i - 1][j - 1] + 1, // substitution
                    matrix[i - 1][j] + 1, // insertion
                    matrix[i][j - 1] + 1 // deletion
                );
            }
        }
    }
}
```

```
        matrix[i][j - 1] + 1,      // insertion
        matrix[i - 1][j] + 1      // deletion
    );
}
}

return matrix[b.length][a.length];
}

// Display search results
function displayResults(searchType, result) {
    resultContainer.innerHTML = '';

    const header = document.createElement('div');
    header.innerHTML = `<h3>${searchType} Search Results</h3>`;
    resultContainer.appendChild(header);

    const timeElement = document.createElement('div');
    timeElement.className = 'search-time';
    timeElement.textContent = `Search time: ${result.time.toFixed(4)} ms`;
    resultContainer.appendChild(timeElement);

    if (result.sortedArray) {
        const sortedElement = document.createElement('div');
        sortedElement.className = 'sorted-array';
        sortedElement.innerHTML = `<strong>Sorted Array:</strong>
${result.sortedArray.join(', ')}`;
        resultContainer.appendChild(sortedElement);
    }

    if (result.found) {
        const resultsElement = document.createElement('div');
        resultsElement.className = 'search-results';

        result.results.forEach(item => {
            const resultItem = document.createElement('div');
            resultItem.className = 'search-result-item';

            let details = `Found at index ${item.index}: ${item.value}
(${item.comparisons} comparisons)`;
            if (item.matchType) {
                details += ` - ${item.matchType} match`;
                if (item.distance !== undefined) {
                    details += ` (distance: ${item.distance})`;
                }
            }
            resultItem.textContent = details;
            resultsElement.appendChild(resultItem);
        });
    }
}

resultContainer.appendChild(resultsElement);
} else {
}
```

```
        const notFoundElement = document.createElement('div');
        notFoundElement.className = 'not-found';
        notFoundElement.textContent = 'Item not found in the array';
        resultContainer.appendChild(notFoundElement);
    }
}

// Linear search button click handler
linearSearchBtn.addEventListener('click', () => {
    safeExecute(() => {
        const data = parseData();
        const query = searchQuery.value.trim();

        if (data.length === 0) {
            throw new Error('Please enter comma-separated data');
        }

        if (query === '') {
            throw new Error('Please enter a search query');
        }

        const result = linearSearch(data, query);
        displayResults('Linear', result);

        return `Linear search complete: ${result.found ? 'Found' : 'Not
found'} `;
    }, error => {
        resultContainer.innerHTML = `<div class="error">${error.message}
</div>`;
    });
});

// Binary search button click handler
binarySearchBtn.addEventListener('click', () => {
    safeExecute(() => {
        const data = parseData();
        const query = searchQuery.value.trim();

        if (data.length === 0) {
            throw new Error('Please enter comma-separated data');
        }

        if (query === '') {
            throw new Error('Please enter a search query');
        }

        const result = binarySearch(data, query);
        displayResults('Binary', result);

        return `Binary search complete: ${result.found ? 'Found' : 'Not
found'} `;
    }, error => {
        resultContainer.innerHTML = `<div class="error">${error.message}
</div>`;
    });
});
```

```
    });

    // Fuzzy search button click handler
    fuzzySearchBtn.addEventListener('click', () => {
        safeExecute(() => {
            const data = parseData();
            const query = searchQuery.value.trim();

            if (data.length === 0) {
                throw new Error('Please enter comma-separated data');
            }

            if (query === '') {
                throw new Error('Please enter a search query');
            }

            const result = fuzzySearch(data, query);
            displayResults('Fuzzy', result);

            return `Fuzzy search complete: ${result.found ? 'Found' : 'Not found'}`;
        }, error => {
            resultContainer.innerHTML = `<div class="error">${error.message}</div>`;
        });
    });
}

// ===== 2.4 Binary Operations =====
function setupBinaryOperations() {
    const binaryInputA = document.getElementById('binary-input-a');
    const binaryInputB = document.getElementById('binary-input-b');
    const andBtn = document.getElementById('binary-and');
    const orBtn = document.getElementById('binary-or');
    const xorBtn = document.getElementById('binary-xor');
    const notBtn = document.getElementById('binary-not');
    const shiftBtn = document.getElementById('binary-shift');
    const resultContainer = document.getElementById('binary-result');

    // Convert number to binary string with padding
    function toBinaryString(number, bits = 8) {
        return (number >>> 0).toString(2).padStart(bits, '0');
    }

    // Display binary representation
    function displayBinary(label, value) {
        const binaryString = toBinaryString(value);

        const container = document.createElement('div');
        container.className = 'binary-display';

        const labelElement = document.createElement('div');
        labelElement.className = 'binary-label';

```

```
labelElement.textContent = label;
container.appendChild(labelElement);

const binaryRow = document.createElement('div');
binaryRow.className = 'binary-row';

// Create bit elements
for (let i = 0; i < binaryString.length; i++) {
    const bit = document.createElement('div');
    bit.className = `bit ${binaryString[i] === '1' ? 'one' : 'zero'}`;
    bit.textContent = binaryString[i];
    binaryRow.appendChild(bit);
}

container.appendChild(binaryRow);

const decimalValue = document.createElement('div');
decimalValue.className = 'decimal-value';
decimalValue.textContent = `Decimal: ${value}`;
container.appendChild(decimalValue);

return container;
}

// Display binary operation result
function displayBinaryOperation(valueA, valueB, result, operationSymbol) {
    resultContainer.innerHTML = '';

    // Display inputs
    resultContainer.appendChild(displayBinary('Value A', valueA));
    if (valueB !== undefined) {
        resultContainer.appendChild(displayBinary('Value B', valueB));
    }

    // Display operation
    const operationElement = document.createElement('div');
    operationElement.className = 'binary-operation';
    operationElement.textContent = operationSymbol;
    resultContainer.appendChild(operationElement);

    // Display result
    resultContainer.appendChild(displayBinary('Result', result));
}

// AND operation
andBtn.addEventListener('click', () => {
    safeExecute(() => {
        const valueA = parseInt(binaryInputA.value) || 0;
        const valueB = parseInt(binaryInputB.value) || 0;

        if (valueA < 0 || valueA > 255 || valueB < 0 || valueB > 255) {
            throw new Error('Values must be between 0 and 255');
    })
})
```

```
        const result = valueA & valueB;
        displayBinaryOperation(valueA, valueB, result, 'AND');

        return `Binary AND: ${valueA} & ${valueB} = ${result}`;
    }, error => {
        resultContainer.innerHTML = `<div class="error">${error.message}</div>`;
    });
});

// OR operation
orBtn.addEventListener('click', () => {
    safeExecute(() => {
        const valueA = parseInt(binaryInputA.value) || 0;
        const valueB = parseInt(binaryInputB.value) || 0;

        if (valueA < 0 || valueA > 255 || valueB < 0 || valueB > 255) {
            throw new Error('Values must be between 0 and 255');
        }

        const result = valueA | valueB;
        displayBinaryOperation(valueA, valueB, result, 'OR');

        return `Binary OR: ${valueA} | ${valueB} = ${result}`;
    }, error => {
        resultContainer.innerHTML = `<div class="error">${error.message}</div>`;
    });
});

// XOR operation
xorBtn.addEventListener('click', () => {
    safeExecute(() => {
        const valueA = parseInt(binaryInputA.value) || 0;
        const valueB = parseInt(binaryInputB.value) || 0;

        if (valueA < 0 || valueA > 255 || valueB < 0 || valueB > 255) {
            throw new Error('Values must be between 0 and 255');
        }

        const result = valueA ^ valueB;
        displayBinaryOperation(valueA, valueB, result, 'XOR');

        return `Binary XOR: ${valueA} ^ ${valueB} = ${result}`;
    }, error => {
        resultContainer.innerHTML = `<div class="error">${error.message}</div>`;
    });
});

// NOT operation
notBtn.addEventListener('click', () => {
    safeExecute(() => {
        const valueA = parseInt(binaryInputA.value) || 0;
```

```
        if (valueA < 0 || valueA > 255) {
            throw new Error('Value must be between 0 and 255');
        }

        const result = ~valueA & 0xFF; // Mask to 8 bits
        displayBinaryOperation(valueA, undefined, result, 'NOT');

        return `Binary NOT: ~${valueA} = ${result}`;
    }, error => {
    resultContainer.innerHTML = `<div class="error">${error.message}
</div>`;
});
});

// Shift Left operation
shiftBtn.addEventListener('click', () => {
safeExecute() => {
    const valueA = parseInt(binaryInputA.value) || 0;
    const shiftAmount = parseInt(binaryInputB.value) || 0;

    if (valueA < 0 || valueA > 255) {
        throw new Error('Value must be between 0 and 255');
    }

    if (shiftAmount < 0 || shiftAmount > 7) {
        throw new Error('Shift amount must be between 0 and 7');
    }

    const result = (valueA << shiftAmount) & 0xFF; // Mask to 8 bits
    displayBinaryOperation(valueA, shiftAmount, result, `SHIFT LEFT
(${shiftAmount} bits)`);

    return `Binary Shift Left: ${valueA} << ${shiftAmount} =
${result}`;
}, error => {
    resultContainer.innerHTML = `<div class="error">${error.message}
</div>`;
});
});

// ===== 3. SYSTEM OPERATIONS =====

// ===== 3.1 File System Operations =====
function setupFileOperations() {
    const fileContent = document.getElementById('file-content');
    const fileName = document.getElementById('file-name');
    const saveFileBtn = document.getElementById('save-file');
    const loadFileBtn = document.getElementById('load-file');
    const fileUpload = document.getElementById('file-upload');
    const createDirBtn = document.getElementById('create-directory');
    const createZipBtn = document.getElementById('create-zip');
    const fileTree = document.getElementById('file-tree');
```

```
console.log("Setting up file operations...");

// Initialize virtual file system if it doesn't exist
if (!window.virtualFS) {
    console.log("Creating new virtual file system");
    window.virtualFS = {
        root: {
            type: 'directory',
            name: 'root',
            children: {}
        }
    };
} else {
    console.log("Using existing virtual file system", window.virtualFS);
}

// Load file button click handler
loadFileBtn.addEventListener('click', () => {
    fileUpload.click();
});

// File upload change handler
fileUpload.addEventListener('change', (event) => {
    safeExecute(() => {
        const file = event.target.files[0];

        if (!file) return;

        const reader = new FileReader();

        reader.onload = (e) => {
            const content = e.target.result;
            fileContent.value = content;
            fileName.value = file.name;

            // Add to virtual filesystem
            addFileToVFS(file.name, content);
            renderFileTree();
        };

        reader.onerror = () => {
            throw new Error('Error reading file');
        };

        reader.readAsText(file);
    }, error => {
        showError(`File loading error: ${error.message}`);
    });
});

// Save file button click handler
saveFileBtn.addEventListener('click', () => {
    safeExecute(() => {
```

```
const content = fileContent.value;
const name = fileName.value.trim();

if (!name) {
    throw new Error('File name cannot be empty');
}

// Save file to browser (download)
const blob = new Blob([content], { type: 'text/plain' });
const url = URL.createObjectURL(blob);
const a = document.createElement('a');
a.href = url;
a.download = name;
a.click();

// Clean up
setTimeout(() => URL.revokeObjectURL(url), 1000);

// Add to virtual filesystem
addFileToVFS(name, content);
renderFileTree();

    return `File "${name}" saved successfully`;
}, error => {
    showError(`File saving error: ${error.message}`);
});
});

// Create directory button click handler
createDirBtn.addEventListener('click', () => {
    safeExecute(() => {
        const dirName = prompt('Enter directory name:');

        if (!dirName) return;

        if (dirName.trim() === '') {
            throw new Error('Directory name cannot be empty');
        }

        addDirectoryToVFS(dirName);
        renderFileTree();

        return `Directory "${dirName}" created`;
}, error => {
    showError(`Directory creation error: ${error.message}`);
});
});

// Create ZIP button click handler (using JSZip)
createZipBtn.addEventListener('click', () => {
    safeExecute(async () => {
        const zip = new JSZip();

        // Add all files from virtual filesystem
```

```
    addFilesToZip(zip, virtualFS.root);

    // Generate ZIP blob
    const blob = await zip.generateAsync({ type: 'blob' });

    // Download ZIP file
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = 'files.zip';
    a.click();

    // Clean up
    setTimeout(() => URL.revokeObjectURL(url), 1000);

    return 'ZIP file created and downloaded';
}, error => {
  showError(`ZIP creation error: ${error.message}`);
});
});

// Helper function to add files to ZIP
function addFilesToZip(zip, directory, path = '') {
  for (const name in directory.children) {
    const item = directory.children[name];
    const itemPath = path ? `${path}/${name}` : name;

    if (item.type === 'file') {
      zip.file(itemPath, item.content);
    } else if (item.type === 'directory') {
      const folder = zip.folder(itemPath);
      addFilesToZip(zip, item, itemPath);
    }
  }
}

// Add file to virtual filesystem
function addFileToVFS(filePath, content) {
  const parts = filePath.split('/');
  const fileName = parts.pop();
  let currentDir = virtualFS.root;

  // Create directories in path if needed
  for (const part of parts) {
    if (!part) continue;

    if (!currentDir.children[part]) {
      currentDir.children[part] = {
        type: 'directory',
        name: part,
        children: {}
      };
    }
  }
}
```

```
        currentDir = currentDir.children[part];
    }

    // Add file
    currentDir.children[fileNames] = {
        type: 'file',
        name: fileName,
        content: content
    };
}

// Add directory to virtual filesystem
function addDirectoryToVFS(dirPath) {
    const parts = dirPath.split('/');
    let currentDir = virtualFS.root;

    // Create directories in path
    for (const part of parts) {
        if (!part) continue;

        if (!currentDir.children[part]) {
            currentDir.children[part] = {
                type: 'directory',
                name: part,
                children: {}
            };
        }
    }

    currentDir = currentDir.children[part];
}
}

// Render file tree
function renderFileTree() {
    fileTree.innerHTML = '';

    function createTreeItem(item, parent) {
        const li = document.createElement('li');

        if (item.type === 'directory') {
            const dirElement = document.createElement('div');
            dirElement.className = 'directory-item';
            dirElement.innerHTML = `<span class="file-icon">📁</span>
${item.name}</span>`;
            li.appendChild(dirElement);

            const childList = document.createElement('ul');
            childList.className = 'directory-children';
        }
    }
}
```

```
        return a.type === 'directory' ? -1 : 1;
    }
    return a.name.localeCompare(b.name);
});

for (const child of children) {
    createTreeItem(child, childList);
}

li.appendChild(childList);

// Toggle expand/collapse
dirElement.addEventListener('click', () => {
    childList.style.display = childList.style.display ===
    'none' ? 'block' : 'none';
});
} else {
    li.className = 'file-item';
    li.innerHTML = `<span class="file-icon">📄</span> <span
class="file-name">${item.name}</span>`;

    // Click to load file
    li.addEventListener('click', () => {
        fileContent.value = item.content;
        fileName.value = item.name;
    });
}
parent.appendChild(li);
}

// Create the root node
createTreeItem(virtualFS.root, fileTree);
}

// Initialize with empty file tree
renderFileTree();
}

// ===== 3.2 Memory Operations =====
function setupMemoryOperations() {
    const memoryFile = document.getElementById('memory-file');
    const memorySlot = document.getElementById('memory-slot');
    const memoryContent = document.getElementById('memory-content');
    const assignSlotBtn = document.getElementById('assign-slot');
    const readSlotBtn = document.getElementById('read-slot');
    const clearAllSlotsBtn = document.getElementById('clear-all-slots');
    const memoryDisplay = document.getElementById('memory-display');

    // Memory manager class
    class MemorySlotManager {
        constructor() {
            this.memorySlots = {};
        }
    }
}
```

```
assignSlot(fileNumber, slotNumber, value) {
    if (!fileNumber || !slotNumber || !value) {
        return "Error: Invalid input parameters.";
    }

    if (this.valueExists(value)) {
        return "Error: The value already exists in another memory
slot.";
    }

    if (!this.memorySlots[fileNumber]) {
        this.memorySlots[fileNumber] = {};
    }

    this.memorySlots[fileNumber][slotNumber] = value;
    return `Assigned value to slot ${slotNumber} in file
${fileNumber}.`;
}

readSlot(fileNumber, slotNumber) {
    if (this.memorySlots[fileNumber] && this.memorySlots[fileNumber]
[slotNumber] !== undefined) {
        return this.memorySlots[fileNumber][slotNumber];
    }
    return `Slot ${slotNumber} not found in file ${fileNumber}.`;
}

valueExists(value) {
    const valueLower = value.toLowerCase();

    for (const file in this.memorySlots) {
        for (const slot in this.memorySlots[file]) {
            if (this.memorySlots[file][slot].toLowerCase() ===
valueLower) {
                return true;
            }
        }
    }
    return false;
}

clearAll() {
    this.memorySlots = {};
    return 'All memory slots cleared';
}
}

// Initialize memory slot manager
const memoryManager = new MemorySlotManager();

// Update memory display
function updateMemoryDisplay() {
```

```
memoryDisplay.innerHTML = ``;

const memorySlots = memoryManager.memorySlots;

if (Object.keys(memorySlots).length === 0) {
    memoryDisplay.innerHTML =
        <div style="text-align: center; color: #888; margin-top:
20px;">
            No memory slots available.
            <br><br>
            Use the controls to add memory slots.
        </div>
    ;
    return;
}

// Sort file numbers numerically if possible
const sortedFiles = Object.keys(memorySlots).sort((a, b) => {
    if (/^\d+$/.test(a) && /^\d+$/.test(b)) {
        return parseInt(a) - parseInt(b);
    }
    return a.localeCompare(b);
});

for (const fileNumber of sortedFiles) {
    const slots = memorySlots[fileNumber];
    const slotCount = Object.keys(slots).length;

    const fileElement = document.createElement('div');
    fileElement.className = 'memory-file';

    const fileTitle = document.createElement('div');
    fileTitle.className = 'memory-file-title';
    fileTitle.textContent = `FILE ${fileNumber} (${slotCount} slots)`;
    fileElement.appendChild(fileTitle);

    // Sort slot numbers numerically if possible
    const sortedSlots = Object.keys(slots).sort((a, b) => {
        if (/^\d+$/.test(a) && /^\d+$/.test(b)) {
            return parseInt(a) - parseInt(b);
        }
        return a.localeCompare(b);
    });

    for (const slotNumber of sortedSlots) {
        const value = slots[slotNumber];

        // Truncate long values for display
        let displayValue = value;
        if (displayValue.length > 100) {
            displayValue = displayValue.substring(0, 97) + '...';
        }

        const slotElement = document.createElement('div');
```

```
slotElement.className = 'memory-slot';

const slotTitle = document.createElement('div');
slotTitle.className = 'memory-slot-title';
slotTitle.textContent = `Slot ${slotNumber}`;

const slotValue = document.createElement('div');
slotValue.className = 'memory-slot-value';
slotValue.textContent = displayName;

slotElement.appendChild(slotTitle);
slotElement.appendChild(slotValue);
fileElement.appendChild(slotElement);
}

memoryDisplay.appendChild(fileElement);
}
}

// Assign slot button click handler
assignSlotBtn.addEventListener('click', () => {
safeExecute(() => {
    const fileNum = memoryFile.value.trim();
    const slotNum = memorySlot.value.trim();
    const content = memoryContent.value;

    if (!fileNum) {
        throw new Error('File number cannot be empty');
    }

    if (!slotNum) {
        throw new Error('Slot number cannot be empty');
    }

    if (!content) {
        throw new Error('Content cannot be empty');
    }

    const result = memoryManager.assignSlot(fileNum, slotNum,
content);

    if (result.startsWith('Error')) {
        throw new Error(result.substring(7));
    }

    updateMemoryDisplay();

    return result;
}, error => {
    showError(`Memory error: ${error.message}`);
});
});

// Read slot button click handler
```

```
readSlotBtn.addEventListener('click', () => {
    safeExecute(() => {
        const fileNum = memoryFile.value.trim();
        const slotNum = memorySlot.value.trim();

        if (!fileNum) {
            throw new Error('File number cannot be empty');
        }

        if (!slotNum) {
            throw new Error('Slot number cannot be empty');
        }

        const result = memoryManager.readSlot(fileNum, slotNum);

        if (result.startsWith('Slot')) {
            throw new Error(result);
        }

        memoryContent.value = result;

        return `Read value from slot ${slotNum} in file ${fileNum}`;
    }, error => {
        showError(`Memory error: ${error.message}`);
    });
});

// Clear all slots button click handler
clearAllSlotsBtn.addEventListener('click', () => {
    if (confirm('Are you sure you want to clear all memory slots?')) {
        memoryManager.clearAll();
        updateMemoryDisplay();
        memoryContent.value = '';
    }
});

// Initialize memory display
updateMemoryDisplay();
}

// ===== 3.3 Process Operations =====
function setupProcessOperations() {
    const processName = document.getElementById('process-name');
    const processDuration = document.getElementById('process-duration');
    const startProcessBtn = document.getElementById('start-process');
    const stopAllProcessesBtn = document.getElementById('stop-all-processes');
    const processList = document.getElementById('process-list');

    // Process manager class
    class ProcessManager {
        constructor() {
            this.processes = new Map();
            this.nextProcessId = 1;
        }
    }
}
```

```
startProcess(name, duration, progressCallback, completeCallback) {
    const processId = this.nextProcessId++;
    const startTime = performance.now();
    const endTime = startTime + duration;

    // Create process object
    const process = {
        id: processId,
        name,
        duration,
        startTime,
        endTime,
        progress: 0,
        status: 'running',
        intervalId: null
    };

    // Start progress updates
    process.intervalId = setInterval(() => {
        const now = performance.now();
        const elapsed = now - startTime;

        if (elapsed >= duration) {
            // Process complete
            process.progress = 100;
            process.status = 'completed';
            clearInterval(process.intervalId);

            // Remove process after a delay
            setTimeout(() => {
                this.processes.delete(processId);
                if (completeCallback) {
                    completeCallback(process);
                }
            }, 1000);
        } else {
            // Update progress
            process.progress = (elapsed / duration) * 100;
        }

        if (progressCallback) {
            progressCallback(process);
        }
    }, 50);

    // Add to process list
    this.processes.set(processId, process);

    return processId;
}

stopProcess(processId) {
    const process = this.processes.get(processId);
```

```
        if (process) {
            clearInterval(process.intervalId);
            process.status = 'stopped';
            this.processes.delete(processId);
            return true;
        }

        return false;
    }

stopAllProcesses() {
    for (const [processId, process] of this.processes.entries()) {
        clearInterval(process.intervalId);
        process.status = 'stopped';
    }

    this.processes.clear();
    return true;
}

// Create process manager
const processManager = new ProcessManager();

// Create process element
function createProcessElement(process) {
    const processElement = document.createElement('div');
    processElement.className = 'process';
    processElement.id = `process-${process.id}`;

    const processHeader = document.createElement('div');
    processHeader.className = 'process-header';

    const processNameElement = document.createElement('div');
    processNameElement.className = 'process-name';
    processNameElement.textContent = process.name;

    const processClose = document.createElement('span');
    processClose.className = 'process-close';
    processClose.textContent = 'x';
    processClose.title = 'Stop process';
    processClose.addEventListener('click', () => {
        processManager.stopProcess(process.id);
        processElement.remove();
    });
}

processHeader.appendChild(processNameElement);
processHeader.appendChild(processClose);

const processProgress = document.createElement('div');
processProgress.className = 'process-progress';

const processBar = document.createElement('div');
```

```
processBar.className = 'process-bar';
processBar.style.width = `${process.progress}%`;

processProgress.appendChild(processBar);

const processStatus = document.createElement('div');
processStatus.className = 'process-status';
processStatus.textContent = `Status: ${process.status} - Progress: ${Math.round(process.progress)}%`;

processElement.appendChild(processHeader);
processElement.appendChild(processProgress);
processElement.appendChild(processStatus);

return processElement;
}

// Update process element
function updateProcessElement(process) {
    const processElement =
document.getElementById(`process-${process.id}`);

    if (processElement) {
        const processBar = processElement.querySelector('.process-bar');
        const processStatus = processElement.querySelector('.process-
status');

        processBar.style.width = `${process.progress}%`;
        processStatus.textContent = `Status: ${process.status} - Progress: ${Math.round(process.progress)}%`;

        if (process.status === 'completed') {
            processElement.classList.add('completed');
        }
    }
}

// Start process button click handler
startProcessBtn.addEventListener('click', () => {
    safeExecute(() => {
        const name = processName.value.trim();
        const duration = parseInt(processDuration.value);

        if (!name) {
            throw new Error('Process name cannot be empty');
        }

        if (isNaN(duration) || duration < 100 || duration > 10000) {
            throw new Error('Duration must be between 100 and 10000 ms');
        }

        // Start the process
        const processId = processManager.startProcess(
            name,
```

```
duration,
updateProcessElement,
process => {
    // Process completed callback
    const processElement =
document.getElementById(`process-${process.id}`);
    if (processElement) {
        setTimeout(() => {
            processElement.classList.add('fade-out');
            setTimeout(() => {
                processElement.remove();
            }, 500);
        }, 1000);
    }
};

// Create and add process element
const process = processManager.processes.get(processId);
const processElement = createProcessElement(process);
processList.appendChild(processElement);

// Clear input
processName.value = '';

    return `Process "${name}" started`;
}, error => {
    showError(`Process error: ${error.message}`);
});
});

// Stop all processes button click handler
stopAllProcessesBtn.addEventListener('click', () => {
    processManager.stopAllProcesses();
    processList.innerHTML = '';
});
}

// ===== 4. CRUD OPERATIONS =====

// Simple in-memory database
const database = {
    records: new Map(),

    create(record) {
        if (!record.id) {
            throw new Error('Record must have an ID');
        }

        if (this.records.has(record.id)) {
            throw new Error(`Record with ID "${record.id}" already exists`);
        }

        this.records.set(record.id, { ...record, createdAt: new Date() });
    }
}
```

```
        return true;
    },

    read(id) {
        if (!this.records.has(id)) {
            throw new Error(`Record with ID "${id}" not found`);
        }

        return { ...this.records.get(id) };
    },

    readAll() {
        return Array.from(this.records.values()).map(record => ({ ...record
}));}
    },

    update(id, updates) {
        if (!this.records.has(id)) {
            throw new Error(`Record with ID "${id}" not found`);
        }

        const record = this.records.get(id);
        const updatedRecord = { ...record, ...updates, updatedAt: new Date()
};}
        this.records.set(id, updatedRecord);

        return updatedRecord;
    },

    delete(id) {
        if (!this.records.has(id)) {
            throw new Error(`Record with ID "${id}" not found`);
        }

        return this.records.delete(id);
    },

    deleteAll() {
        this.records.clear();
        return true;
    },

    query(criteria) {
        return Array.from(this.records.values())
            .filter(record => {
                for (const key in criteria) {
                    if (record[key] !== criteria[key]) {
                        return false;
                    }
                }
                return true;
            })
            .map(record => ({ ...record }));
    }
}
```

```
};

// ===== 4.1 Create Operations =====
function setupCreateOperations() {
    const recordId = document.getElementById('record-id');
    const recordName = document.getElementById('record-name');
    const recordData = document.getElementById('record-data');
    const createRecordBtn = document.getElementById('create-record');
    const databaseDisplay = document.getElementById('database-display');

    // Create record button click handler
    createRecordBtn.addEventListener('click', () => {
        safeExecute(() => {
            const id = recordId.value.trim();
            const name = recordName.value.trim();
            const data = recordData.value.trim();

            if (!id) {
                throw new Error('Record ID cannot be empty');
            }

            if (!name) {
                throw new Error('Record name cannot be empty');
            }

            // Create the record
            database.create({
                id,
                name,
                data
            });

            // Clear inputs
            recordId.value = '';
            recordName.value = '';
            recordData.value = '';

            // Update the display
            updateDatabaseDisplay();

            return `Record "${id}" created successfully`;
        }, error => {
            showError(`Create error: ${error.message}`);
        });
    });
}

// Update database display
function updateDatabaseDisplay() {
    databaseDisplay.innerHTML = '';

    const records = database.readAll();

    if (records.length === 0) {
        databaseDisplay.innerHTML = `
```

```
<div style="text-align: center; color: #888; margin-top: 20px;">
    No records in the database.
    <br><br>
    Use the form to create records.
</div>
`;
return;
}

// Sort records by ID
records.sort((a, b) => a.id.localeCompare(b.id));

for (const record of records) {
    const recordElement = document.createElement('div');
    recordElement.className = 'database-record';

    const recordHeader = document.createElement('div');
    recordHeader.className = 'record-header';

    const recordIdElement = document.createElement('div');
    recordIdElement.className = 'record-id';
    recordIdElement.textContent = record.id;

    const recordControls = document.createElement('div');
    recordControls.className = 'record-controls';

    recordHeader.appendChild(recordIdElement);
    recordHeader.appendChild(recordControls);

    const recordContent = document.createElement('div');
    recordContent.className = 'record-content';
    recordContent.innerHTML =
        <div><strong>Name:</strong> ${record.name}</div>
        <div><strong>Data:</strong> ${record.data || '<empty>'}</div>
        <div><strong>Created:</strong>
${record.createdAt.toLocaleString()}</div>
        ${record.updatedAt ? `<div><strong>Updated:</strong>
${record.updatedAt.toLocaleString()}</div>` : ''}
    `;

    recordElement.appendChild(recordHeader);
    recordElement.appendChild(recordContent);

    databaseDisplay.appendChild(recordElement);
}

// Initialize the database display
updateDatabaseDisplay();

// Export for other functions
return { updateDatabaseDisplay };
}
```

```
// ===== 4.2 Read Operations =====
function setupReadOperations(displayUpdater) {
    const readId = document.getElementById('read-id');
    const readRecordBtn = document.getElementById('read-record');
    const readAllRecordsBtn = document.getElementById('read-all-records');
    const queryRecordsBtn = document.getElementById('query-records');
    const readResult = document.getElementById('read-result');

    // Read record button click handler
    readRecordBtn.addEventListener('click', () => {
        safeExecute(() => {
            const id = readId.value.trim();

            if (!id) {
                throw new Error('Record ID cannot be empty');
            }

            const record = database.read(id);

            // Display the record
            readResult.innerHTML = '';

            const recordElement = document.createElement('div');
            recordElement.className = 'database-record';

            const recordHeader = document.createElement('div');
            recordHeader.className = 'record-header';

            const recordIdElement = document.createElement('div');
            recordIdElement.className = 'record-id';
            recordIdElement.textContent = record.id;

            recordHeader.appendChild(recordIdElement);

            const recordContent = document.createElement('div');
            recordContent.className = 'record-content';
            recordContent.innerHTML =
                `<div><strong>Name:</strong> ${record.name}</div>
                 <div><strong>Data:</strong> ${record.data || '<empty>'}</div>
                 <div><strong>Created:</strong>
${record.createdAt.toLocaleString()}</div>
                 ${record.updatedAt ? `<div><strong>Updated:</strong>
${record.updatedAt.toLocaleString()}</div>` : ''}
            `;

            recordElement.appendChild(recordHeader);
            recordElement.appendChild(recordContent);

            readResult.appendChild(recordElement);

            return `Record "${id}" read successfully`;
        }, error => {
            readResult.innerHTML = `<div class="error">${error.message}`

    
```

```
</div>`;
    });
});

// Read all records button click handler
readAllRecordsBtn.addEventListener('click', () => {
    safeExecute(() => {
        const records = database.readAll();

        if (records.length === 0) {
            readResult.innerHTML =
                `<div style="text-align: center; color: #888; margin-top: 20px;">
                    No records in the database.
                </div>
            `;
            return `No records found in the database`;
        }

        // Display all records
        readResult.innerHTML = '';

        // Sort records by ID
        records.sort((a, b) => a.id.localeCompare(b.id));

        for (const record of records) {
            const recordElement = document.createElement('div');
            recordElement.className = 'database-record';

            const recordHeader = document.createElement('div');
            recordHeader.className = 'record-header';

            const recordIdElement = document.createElement('div');
            recordIdElement.className = 'record-id';
            recordIdElement.textContent = record.id;

            recordHeader.appendChild(recordIdElement);

            const recordContent = document.createElement('div');
            recordContent.className = 'record-content';
            recordContent.innerHTML =
                `<div><strong>Name:</strong> ${record.name}</div>
                <div><strong>Data:</strong> ${record.data || '<empty>'}</div>
            `;
            recordElement.appendChild(recordHeader);
            recordElement.appendChild(recordContent);
        }

        readResult.appendChild(recordElement);
    });
});
```

```
        }

        return `${records.length} records read successfully`;
    }, error => {
        readResult.innerHTML = `<div class="error">${error.message}</div>`;
    });
});

// Query records button click handler
queryRecordsBtn.addEventListener('click', () => {
    safeExecute(() => {
        // Create a query dialog
        const queryValue = prompt('Enter search term to query records by name:');
        if (queryValue === null) return; // User cancelled

        const query = queryValue.trim();

        if (!query) {
            throw new Error('Search term cannot be empty');
        }

        // Simple case-insensitive name search
        const records = database.readAll().filter(record =>
            record.name.toLowerCase().includes(query.toLowerCase())
        );

        if (records.length === 0) {
            readResult.innerHTML =
                `<div style="text-align: center; color: #888; margin-top: 20px;">
                    No records found matching the query "${query}".
                </div>
            `;
            return `No records found matching the query "${query}"`;
        }

        // Display all matching records
        readResult.innerHTML = '';

        // Sort records by ID
        records.sort((a, b) => a.id.localeCompare(b.id));

        for (const record of records) {
            const recordElement = document.createElement('div');
            recordElement.className = 'database-record';

            const recordHeader = document.createElement('div');
            recordHeader.className = 'record-header';

            const recordIdElement = document.createElement('div');
            recordIdElement.className = 'record-id';
        }
    });
});
```

```
    recordIdElement.textContent = record.id;

    recordHeader.appendChild(recordIdElement);

    const recordContent = document.createElement('div');
    recordContent.className = 'record-content';
    recordContent.innerHTML =
        `<div><strong>Name:</strong> ${record.name}</div>
         <div><strong>Data:</strong> ${record.data || '<empty>'}
</div>
         <div><strong>Created:</strong>
${record.createdAt.toLocaleString()}</div>
         ${record.updatedAt ? `<div><strong>Updated:</strong>
${record.updatedAt.toLocaleString()}</div>` : ''}
         `;

    recordElement.appendChild(recordHeader);
    recordElement.appendChild(recordContent);

    readResult.appendChild(recordElement);
}

return `${records.length} records found matching the query
"${query}"`;
}, error => {
    readResult.innerHTML = `<div class="error">${error.message}
</div>`;
});
});
}
}

// ===== 4.3 Update Operations =====
function setupUpdateOperations(displayUpdater) {
    const updateId = document.getElementById('update-id');
    const updateField = document.getElementById('update-field');
    const updateValue = document.getElementById('update-value');
    const updateRecordBtn = document.getElementById('update-record');
    const updateResult = document.getElementById('update-result');

    // Update record button click handler
    updateRecordBtn.addEventListener('click', () => {
        safeExecute(() => {
            const id = updateId.value.trim();
            const field = updateField.value;
            const value = updateValue.value.trim();

            if (!id) {
                throw new Error('Record ID cannot be empty');
            }

            if (!field) {
                throw new Error('Field cannot be empty');
            }
        });
    });
}
```

```
// Create update object
const updates = {
    [field]: value
};

// Update the record
const updatedRecord = database.update(id, updates);

// Display the updated record
updateResult.innerHTML = `

    const recordElement = document.createElement('div');
    recordElement.className = 'database-record';

    const recordHeader = document.createElement('div');
    recordHeader.className = 'record-header';

    const recordIdElement = document.createElement('div');
    recordIdElement.className = 'record-id';
    recordIdElement.textContent = updatedRecord.id;

    recordHeader.appendChild(recordIdElement);

    const recordContent = document.createElement('div');
    recordContent.className = 'record-content';
    recordContent.innerHTML =
        <div><strong>Name:</strong> ${updatedRecord.name}</div>
        <div><strong>Data:</strong> ${updatedRecord.data || '<empty>'}
</div>
        <div><strong>Created:</strong>
${updatedRecord.createdAt.toLocaleString()}</div>
            <div><strong>Updated:</strong>
${updatedRecord.updatedAt.toLocaleString()}</div>
    `;

    recordElement.appendChild(recordHeader);
    recordElement.appendChild(recordContent);

    updateResult.appendChild(recordElement);

    // Clear inputs
    updateId.value = '';
    updateValue.value = '';

    // Update the database display
    if (displayUpdater) displayUpdater.updateDatabaseDisplay();

    return `Record "${id}" updated successfully`;
}, error => {
    updateResult.innerHTML = `<div class="error">${error.message}
</div>`;
});`);
}
```

```
// ===== 4.4 Delete Operations =====
function setupDeleteOperations(displayUpdater) {
    const deleteId = document.getElementById('delete-id');
    const deleteRecordBtn = document.getElementById('delete-record');
    const deleteAllRecordsBtn = document.getElementById('delete-all-records');
    const deleteResult = document.getElementById('delete-result');

    // Delete record button click handler
    deleteRecordBtn.addEventListener('click', () => {
        safeExecute(() => {
            const id = deleteId.value.trim();

            if (!id) {
                throw new Error('Record ID cannot be empty');
            }

            // Delete the record
            const result = database.delete(id);

            if (!result) {
                throw new Error(`Failed to delete record "${id}"`);
            }

            // Clear input
            deleteId.value = '';

            // Update the database display
            if (displayUpdater) displayUpdater.updateDatabaseDisplay();

            deleteResult.innerHTML = `<div class="status-message status-success">Record "${id}" deleted successfully</div>`;

            return `Record "${id}" deleted successfully`;
        }, error => {
            deleteResult.innerHTML = `<div class="error">${error.message}</div>`;
        });
    });

    // Delete all records button click handler
    deleteAllRecordsBtn.addEventListener('click', () => {
        safeExecute(() => {
            if (!confirm('Are you sure you want to delete all records? This cannot be undone.')) {
                return;
            }

            // Delete all records
            database.deleteAll();

            // Update the database display
            if (displayUpdater) displayUpdater.updateDatabaseDisplay();
        });
    });
}
```

```
        deleteResult.innerHTML = `<div class="status-message status-success">All records deleted successfully</div>`;

        return 'All records deleted successfully';
    }, error => {
        deleteResult.innerHTML = `<div class="error">${error.message}</div>`;
    });
});

}

// ===== 5. ADVANCED CONCEPTS =====

// ===== 5.1 Serialization Operations =====
function setupSerializationOperations() {
    const serializeInput = document.getElementById('serialize-input');
    const serializationFormat = document.getElementById('serialization-format');

    const serializeBtn = document.getElementById('serialize-data');
    const deserializeBtn = document.getElementById('deserialize-data');
    const resultContainer = document.getElementById('serialization-result');

    // Currently serialized data
    let serializedData = '';
    let originalObject = null;

    // Serialize button click handler
    serializeBtn.addEventListener('click', () => {
        safeExecute(() => {
            const input = serializeInput.value.trim();
            const format = serializationFormat.value;

            if (!input) {
                throw new Error('Input cannot be empty');
            }

            // Parse the input as JSON to get an object
            try {
                originalObject = JSON.parse(input);
            } catch (e) {
                throw new Error('Invalid JSON input. Please check your syntax.');
            }

            // Serialize the object according to the selected format
            serializedData = serializeObject(originalObject, format);

            // Display the serialized data
            resultContainer.innerHTML = '';

            const resultHeader = document.createElement('div');
            resultHeader.className = 'serialization-header';
            resultHeader.innerHTML = `<strong>Serialized Data (${format}):</strong>`;
        });
    });
}
```

```
resultContainer.appendChild(resultHeader);

const resultContent = document.createElement('pre');
resultContent.className = 'serialization-content';
resultContent.textContent = serializedData;
resultContainer.appendChild(resultContent);

return `Data serialized to ${format} format`;
}, error => {
  resultContainer.innerHTML = `<div class="error">${error.message}</div>`;
});
});

// Deserialize button click handler
deserializeBtn.addEventListener('click', () => {
  safeExecute(() => {
    const format = serializationFormat.value;

    if (!serializedData) {
      throw new Error('No serialized data to deserialize');
    }

    // Deserialize the data according to the selected format
    const deserializedObject = deserializeData(serializedData,
format);

    // Display the deserialized object
    resultContainer.innerHTML = '';

    const resultHeader = document.createElement('div');
    resultHeader.className = 'serialization-header';
    resultHeader.innerHTML = `<strong>Deserialized Object:</strong>`;
    resultContainer.appendChild(resultHeader);

    const resultContent = document.createElement('pre');
    resultContent.className = 'serialization-content';
    resultContent.textContent = JSON.stringify(deserializedObject,
null, 2);
    resultContainer.appendChild(resultContent);

    // Display comparison
    const comparisonHeader = document.createElement('div');
    comparisonHeader.className = 'serialization-header mt-20';
    comparisonHeader.innerHTML = `<strong>Comparison with Original:</strong>`;
    resultContainer.appendChild(comparisonHeader);

    const isEqual = compareObjects(originalObject,
deserializedObject);

    const comparisonResult = document.createElement('div');
    comparisonResult.className = isEqual ? 'status-success' : 'status-
error';
  });
});
```

```
        comparisonResult.textContent = isEqual
            ? 'Objects are identical. Serialization and deserialization
completed successfully.'
            : 'Objects are different. Some data may have been lost in the
process.';

        resultContainer.appendChild(comparisonResult);

        return `Data deserialized from ${format} format`;
}, error => {
    resultContainer.innerHTML = `<div class="error">${error.message}
</div>`;
});

// Serialize an object to the specified format
function serializeObject(obj, format) {
    switch (format) {
        case 'json':
            return JSON.stringify(obj, null, 2);

        case 'xml':
            return objectToXML(obj);

        case 'csv':
            return objectToCSV(obj);

        default:
            throw new Error(`Unsupported format: ${format}`);
    }
}

// Deserialize data from the specified format
function deserializeData(data, format) {
    switch (format) {
        case 'json':
            return JSON.parse(data);

        case 'xml':
            return xmlToObject(data);

        case 'csv':
            return csvToObject(data);

        default:
            throw new Error(`Unsupported format: ${format}`);
    }
}

// Convert object to XML
function objectToXML(obj) {
    function addIndent(level) {
        return ' '.repeat(level * 2);
    }

    const result = obj
        .map((item, index) => {
            if (Array.isArray(item)) {
                return addIndent(level + 1) + item.map(subItem => objectToXML(subItem)).join('');
            } else {
                return addIndent(level) + `<${item.name} ${item.attributes ? item.attributes : ''}>${item.value}</${item.name}>`;
            }
        })
        .join('');

    return result;
}
```

```
function objectToXMLRecursive(obj, name, level = 0) {
    let xml = `${addIndent(level)}<${name}>`;

    if (obj === null) {
        return `${xml}null</${name}>`;
    }

    if (typeof obj !== 'object') {
        return `${xml}${String(obj)}</${name}>`;
    }

    xml += '\n';

    if (Array.isArray(obj)) {
        for (let i = 0; i < obj.length; i++) {
            xml += objectToXMLRecursive(obj[i], 'item', level + 1) +
'\n';
        }
    } else {
        for (const key in obj) {
            if (obj.hasOwnProperty(key)) {
                xml += objectToXMLRecursive(obj[key], key, level + 1) +
'\n';
            }
        }
    }

    xml += `${addIndent(level)}</${name}>`;
    return xml;
}

return objectToXMLRecursive(obj, 'root');
}

// Convert XML to object (simplified)
function xmlToObject(xml) {
    // This is a very simplified XML parser
    // In a real-world scenario, you would use a proper XML parser

    // Extract content within root tags
    const rootMatch = xml.match(/<root>([\s\S]*?)</root>/);
    if (!rootMatch) {
        throw new Error('Invalid XML: root tag not found');
    }

    const rootContent = rootMatch[1];

    // Parse key-value pairs
    const result = {};
    const keyPattern = /<(\w+)>([\s\S]*?)<\/\1>/g;
    let match;

    while ((match = keyPattern.exec(rootContent)) !== null) {
        const key = match[1];
```

```
const value = match[2].trim();

// Try to parse nested objects
if (value.includes('<')) {
    // Check if it's an array of items
    if (value.includes('<item>')) {
        result[key] = [];
        const itemPattern = /<item>([\s\S]*?)</item>/g;
        let itemMatch;

        while ((itemMatch = itemPattern.exec(value)) !== null) {
            const itemValue = itemMatch[1].trim();

            if (itemValue.includes('<')) {
                // Try to parse as object
                try {
                    const dummyXml = `<root>${itemValue}</root>`;
                    result[key].push(xmlToObject(dummyXml));
                } catch (e) {
                    result[key].push(itemValue);
                }
            } else {
                // Try to parse as number or boolean
                result[key].push(parseValue(itemValue));
            }
        }
    } else {
        // Try to parse as object
        try {
            const dummyXml = `<root>${value}</root>`;
            result[key] = xmlToObject(dummyXml);
        } catch (e) {
            result[key] = value;
        }
    }
} else {
    // Parse as primitive value
    result[key] = parseValue(value);
}

return result;
}

// Convert object to CSV (for flat objects or arrays of flat objects)
function objectToCSV(obj) {
    if (Array.isArray(obj)) {
        // Array of objects - create CSV with headers
        if (obj.length === 0) {
            return '';
        }

        // Get headers from the first object
        const headers = Object.keys(obj[0]);
    }
}
```

```
let csv = headers.join(',') + '\n';

    // Add rows
    for (const item of obj) {
        const row = headers.map(header => {
            const value = item[header];
            return escapeCSV(value);
        }).join(',');
    }

        csv += row + '\n';
    }

    return csv;
} else {
    // Single object - create key-value CSV
    let csv = 'key,value\n';

    for (const key in obj) {
        if (obj.hasOwnProperty(key)) {
            const value = obj[key];
            csv += `${escapeCSV(key)},${escapeCSV(value)}\n`;
        }
    }

    return csv;
}

// Convert CSV to object
function csvToObject(csv) {
    const lines = csv.split('\n').filter(line => line.trim() !== '');
    if (lines.length < 2) {
        return {};
    }

    const headers = lines[0].split(',');

    // Check if it's key-value format
    if (headers.length === 2 && headers[0] === 'key' && headers[1] ===
'value') {
        const result = {};

        for (let i = 1; i < lines.length; i++) {
            const [key, value] = parseCSVLine(lines[i]);
            result[key] = parseValue(value);
        }

        return result;
    } else {
        // Array of objects format
        const result = [];

        for (let i = 1; i < lines.length; i++) {
            const values = parseCSVLine(lines[i]);
            result.push(values);
        }
    }
}
```

```
        const item = {};

        for (let j = 0; j < headers.length; j++) {
            item[headers[j]] = parseValue(values[j] || '');
        }

        result.push(item);
    }

    return result;
}

}

// Parse a CSV line, handling quoted values
function parseCSVLine(line) {
    const values = [];
    let current = '';
    let inQuotes = false;

    for (let i = 0; i < line.length; i++) {
        const char = line[i];

        if (char === '"') {
            if (i + 1 < line.length && line[i + 1] === '"') {
                // Escaped quote
                current += '"';
                i++;
            } else {
                // Toggle quote mode
                inQuotes = !inQuotes;
            }
        } else if (char === ',' && !inQuotes) {
            // End of value
            values.push(current);
            current = '';
        } else {
            current += char;
        }
    }

    // Add the last value
    values.push(current);

    return values;
}

// Escape a value for CSV
function escapeCSV(value) {
    if (value == null) {
        return '';
    }

    const strValue = String(value);
```

```
        if (strValue.includes(',') || strValue.includes('"') ||  
strValue.includes('\n')) {  
            // Escape quotes by doubling them and wrap in quotes  
            return `#${strValue.replace(/\"/g, '\"'')}#`;  
        }  
  
        return strValue;  
    }  
  
    // Parse a value to the appropriate type  
    function parseValue(value) {  
        if (value === 'null') {  
            return null;  
        }  
  
        if (value === 'true') {  
            return true;  
        }  
  
        if (value === 'false') {  
            return false;  
        }  
  
        if (!isNaN(Number(value))) {  
            return Number(value);  
        }  
  
        return value;  
    }  
  
    // Compare two objects for equality  
    function compareObjects(a, b) {  
        if (a === b) {  
            return true;  
        }  
  
        if (typeof a !== 'object' || typeof b !== 'object' || a == null || b  
== null) {  
            return false;  
        }  
  
        const keysA = Object.keys(a);  
        const keysB = Object.keys(b);  
  
        if (keysA.length !== keysB.length) {  
            return false;  
        }  
  
        for (const key of keysA) {  
            if (!keysB.includes(key)) {  
                return false;  
            }  
  
            if (!compareObjects(a[key], b[key])) {  
                return false;  
            }  
        }  
    }  
}
```

```
        return false;
    }
}

return true;
}
}

// ===== 5.2 Compression Operations =====
function setupCompressionOperations() {
    const compressionInput = document.getElementById('compression-input');
    const compressionMethod = document.getElementById('compression-method');
    const compressBtn = document.getElementById('compress-data');
    const decompressBtn = document.getElementById('decompress-data');
    const resultContainer = document.getElementById('compression-result');
    const statsContainer = document.getElementById('compression-stats');

    // Currently compressed data
    let compressedData = '';
    let originalData = '';
    let currentMethod = '';

    // Compress button click handler
    compressBtn.addEventListener('click', () => {
        safeExecute(() => {
            originalData = compressionInput.value;
            currentMethod = compressionMethod.value;

            if (!originalData) {
                throw new Error('Input cannot be empty');
            }

            // Compress the data according to the selected method
            const compressionResult = compressData(originalData,
currentMethod);
            compressedData = compressionResult.data;

            // Display the compressed data
            resultContainer.innerHTML = '';

            const resultHeader = document.createElement('div');
            resultHeader.className = 'compression-header';
            resultHeader.innerHTML = `<strong>Compressed Data
(${currentMethod}):</strong>`;
            resultContainer.appendChild(resultHeader);

            const resultContent = document.createElement('pre');
            resultContent.className = 'compression-content';
            resultContent.textContent = compressedData;
            resultContainer.appendChild(resultContent);

            // Display compression stats
            statsContainer.innerHTML = '';
        });
    });
}
```

```
const originalSize = new Blob([originalData]).size;
const compressedSize = new Blob([compressedData]).size;
const compressionRatio = originalSize / compressedSize;
const spaceSavings = (1 - (compressedSize / originalSize)) * 100;

statsContainer.innerHTML =
    <div><strong>Original Size:</strong> ${originalSize}
bytes</div>
    <div><strong>Compressed Size:</strong> ${compressedSize}
bytes</div>
    <div><strong>Compression Ratio:</strong>
${compressionRatio.toFixed(2)}:1</div>
        <div><strong>Space Savings:</strong>
${spaceSavings.toFixed(2)}%</div>
`;

return `Data compressed using ${currentMethod}`;
}, error => {
    resultContainer.innerHTML = `<div class="error">${error.message}</div>`;
    statsContainer.innerHTML = '';
});
});

// Decompress button click handler
decompressBtn.addEventListener('click', () => {
    safeExecute(() => {
        if (!compressedData) {
            throw new Error('No compressed data to decompress');
        }

        // Decompress the data
        const decompressedData = decompressData(compressedData,
currentMethod);

        // Display the decompressed data
        resultContainer.innerHTML = '';

        const resultHeader = document.createElement('div');
        resultHeader.className = 'compression-header';
        resultHeader.innerHTML = `<strong>Decompressed Data:</strong>`;
        resultContainer.appendChild(resultHeader);

        const resultContent = document.createElement('pre');
        resultContent.className = 'compression-content';
        resultContent.textContent = decompressedData;
        resultContainer.appendChild(resultContent);

        // Update stats
        statsContainer.innerHTML += `
<div class="mt-10"><strong>Verification:</strong> ${
            decompressedData === originalData
                ? '<span class="status-success">Data matches
original</span>'
                : '<span class="status-error">Data does not match
original</span>'`;
```

```
        : '<span class="status-error">Data does not match
original</span>'
    }</div>
    `;

        return `Data decompressed using ${currentMethod}`;
}, error => {
    resultContainer.innerHTML = `<div class="error">${error.message}
</div>`;
})];
});

// Compress data using the specified method
function compressData(data, method) {
    switch (method) {
        case 'rle':
            return { data: runLengthEncode(data) };

        case 'huffman':
            return { data: huffmanEncode(data) };

        case 'lz':
            return { data: lzCompress(data) };

        default:
            throw new Error(`Unsupported compression method: ${method}`);
    }
}

// Decompress data using the specified method
function decompressData(data, method) {
    switch (method) {
        case 'rle':
            return runLengthDecode(data);

        case 'huffman':
            return huffmanDecode(data);

        case 'lz':
            return lzDecompress(data);

        default:
            throw new Error(`Unsupported compression method: ${method}`);
    }
}

// Run-Length Encoding
function runLengthEncode(data) {
    let encoded = '';
    let count = 1;

    for (let i = 0; i < data.length; i++) {
        // If the current character is the same as the next, increment
        count
    }
}
```

```
        if (data[i] === data[i + 1]) {
            count++;
        } else {
            // Otherwise, add the count and character to the result
            if (count >= 4) {
                // Only use RLE for runs of 4 or more
                encoded += `${count}${data[i]}`;
            } else {
                // Otherwise, just use the raw characters
                encoded += data[i].repeat(count);
            }
            count = 1;
        }
    }

    return encoded;
}

// Run-Length Decoding
function runLengthDecode(data) {
    let decoded = '';
    let countStr = '';

    for (let i = 0; i < data.length; i++) {
        const char = data[i];

        if (/^\d/.test(char)) {
            // If it's a digit, add it to the count
            countStr += char;
        } else {
            // Otherwise, it's a character
            let count = countStr ? parseInt(countStr) : 1;
            decoded += char.repeat(count);
            countStr = '';
        }
    }

    return decoded;
}

// Huffman Encoding (simplified version)
function huffmanEncode(data) {
    // Count frequency of each character
    const frequencies = {};
    for (const char of data) {
        frequencies[char] = (frequencies[char] || 0) + 1;
    }

    // Build Huffman tree (simplified as a dictionary)
    // In a real Huffman implementation, we would build a tree and
traverse it
    const codes = {};
    let index = 0;
```

```
for (const char in frequencies) {
    // Use binary representation of index as the code (simpler than a
    real Huffman tree)
    codes[char] = index.toString(2).padStart(4, '0');
    index++;

    if (index >= 16) {
        break; // Limit to 16 distinct characters for simplicity
    }
}

// Encode the data
let encoded = '';

// First, add the dictionary
for (const char in codes) {
    const charCode = char.charCodeAt(0).toString(16).padStart(4, '0');
    encoded += `${charCode}:${codes[char]}`;
}

encoded += '|'; // Dictionary/data separator

// Then encode the data
for (const char of data) {
    encoded += codes[char] || '';
}

return encoded;
}

// Huffman Decoding (simplified version)
function huffmanDecode(data) {
    // Split into dictionary and encoded data
    const parts = data.split('|');
    if (parts.length !== 2) {
        throw new Error('Invalid Huffman encoded data');
    }

    const dictionaryStr = parts[0];
    const encodedData = parts[1];

    // Parse the dictionary
    const codeToChar = {};
    const dictionaryEntries = dictionaryStr.split(';');

    for (const entry of dictionaryEntries) {
        if (!entry) continue;

        const [charHex, code] = entry.split(':');
        if (!charHex || !code) continue;

        const char = String.fromCharCode(parseInt(charHex, 16));
        codeToChar[code] = char;
    }
}
```

```
// Decode the data
let decoded = '';
let currentCode = '';

for (const bit of encodedData) {
    currentCode += bit;

    if (codeToChar[currentCode]) {
        decoded += codeToChar[currentCode];
        currentCode = '';
    }
}

return decoded;
}

// LZ Compression (simplified version of LZ77)
function lzCompress(data) {
    let compressed = '';
    let pos = 0;

    while (pos < data.length) {
        // Look for longest match in the window
        let bestLength = 0;
        let bestOffset = 0;
        const maxOffset = Math.min(255, pos); // Limit to 255 for
simplicity
        const maxLength = Math.min(255, data.length - pos); // Limit to
255 for simplicity

        for (let offset = 1; offset <= maxOffset; offset++) {
            let length = 0;
            while (length < maxLength && data[pos - offset + (length %
offset)] === data[pos + length]) {
                length++;
            }

            if (length > bestLength) {
                bestLength = length;
                bestOffset = offset;
            }
        }

        if (bestLength >= 3) { // Only compress sequences of 3 or more
// Add a reference as (offset, length)
compressed += `(${bestOffset},${bestLength})`;
pos += bestLength;
} else {
    // Add the character as is
    if (/[\\\()]/.test(data[pos])) {
        compressed += '\\\\' + data[pos]; // Escape special
characters
    } else {

```

```
        compressed += data[pos];
    }
    pos++;
}
}

return compressed;
}

// LZ Decompression
function lzDecompress(data) {
    let decompressed = '';
    let pos = 0;

    while (pos < data.length) {
        if (data[pos] === '\\') {
            // Escaped character
            decompressed += data[pos + 1];
            pos += 2;
        } else if (data[pos] === '(') {
            // Reference (offset, length)
            const match = data.substring(pos).match(/^\((\d+),(\d+)\)\)/);
            if (!match) {
                throw new Error('Invalid LZ compressed data');
            }

            const offset = parseInt(match[1]);
            const length = parseInt(match[2]);

            for (let i = 0; i < length; i++) {
                decompressed += decompressed[decompressed.length - offset
+ (i % offset)];
            }
        }

        pos += match[0].length;
    } else {
        // Literal character
        decompressed += data[pos];
        pos++;
    }
}

return decompressed;
}

// ===== 5.3 Binary Encoding Operations =====
function setupEncodingOperations() {
    const encodingInput = document.getElementById('encoding-input');
    const encodingMethod = document.getElementById('encoding-method');
    const encodeBtn = document.getElementById('encode-data');
    const decodeBtn = document.getElementById('decode-data');
    const resultContainer = document.getElementById('encoding-result');
```

```
// Currently encoded data
let encodedData = '';
let originalData = '';
let currentMethod = '';

// Encode button click handler
encodeBtn.addEventListener('click', () => {
    safeExecute(() => {
        originalData = encodingInput.value;
        currentMethod = encodingMethod.value;

        if (!originalData) {
            throw new Error('Input cannot be empty');
        }

        // Encode the data according to the selected method
        encodedData = encodeData(originalData, currentMethod);

        // Display the encoded data
        resultContainer.innerHTML = '';

        const resultHeader = document.createElement('div');
        resultHeader.className = 'encoding-header';
        resultHeader.innerHTML = `<strong>Encoded Data (${currentMethod}):` +
        </strong>`;
        resultContainer.appendChild(resultHeader);

        const resultContent = document.createElement('pre');
        resultContent.className = 'encoding-content';
        resultContent.textContent = encodedData;
        resultContainer.appendChild(resultContent);

        return `Data encoded using ${currentMethod}`;
    }, error => {
        resultContainer.innerHTML = `<div class="error">${error.message}` +
        </div>`;
    });
});

// Decode button click handler
decodeBtn.addEventListener('click', () => {
    safeExecute(() => {
        const input = encodingInput.value;
        currentMethod = encodingMethod.value;

        if (!input) {
            throw new Error('Input cannot be empty');
        }

        // Decode the data
        const decodedData = decodeData(input, currentMethod);

        // Display the decoded data
        resultContainer.innerHTML = '';
    });
});
```

```
const resultHeader = document.createElement('div');
resultHeader.className = 'encoding-header';
resultHeader.innerHTML = `<strong>Decoded Data:</strong>`;
resultContainer.appendChild(resultHeader);

const resultContent = document.createElement('pre');
resultContent.className = 'encoding-content';
resultContent.textContent = decodedData;
resultContainer.appendChild(resultContent);

return `Data decoded from ${currentMethod}`;
}, error => {
    resultContainer.innerHTML = `<div class="error">${error.message}
</div>`;
});
});

// Encode data using the specified method
function encodeData(data, method) {
    switch (method) {
        case 'base64':
            return btoa(unescape(encodeURIComponent(data)));

        case 'hex':
            return [...data].map(c =>
c.charCodeAt(0).toString(16).padStart(2, '0')).join('');

        case 'binary':
            return [...data].map(c =>
c.charCodeAt(0).toString(2).padStart(8, '0')).join(' ');

        case 'custom':
            return generateCustomId(data);

        default:
            throw new Error(`Unsupported encoding method: ${method}`);
    }
}

// Decode data using the specified method
function decodeData(data, method) {
    switch (method) {
        case 'base64':
            return decodeURIComponent(escape(atob(data)));

        case 'hex':
            if (!/^[\0-\xFF]+$/ .test(data) || data.length % 2 !== 0) {
                throw new Error('Invalid hexadecimal string');
            }
            let result = '';
            for (let i = 0; i < data.length; i += 2) {
                result += String.fromCharCode(parseInt(data.substr(i, 2),
16));
            }
    }
}
```

```
        }

        return result;

    case 'binary':
        return data.split(' ')
            .filter(binary => binary.trim() !== '')
            .map(binary => String.fromCharCode(parseInt(binary, 2)))
            .join('');

    case 'custom':
        return decodeCustomId(data);

    default:
        throw new Error(`Unsupported encoding method: ${method}`);
    }
}

// Generate a custom string ID
function generateCustomId(data) {
    const charset =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    const charsetLength = charset.length;

    let id = '';

    // Convert string to a large integer
    for (let i = 0; i < data.length; i++) {
        const charCode = data.charCodeAt(i);
        id = id * charsetLength + (charCode % charsetLength);
    }

    // Add a simple checksum
    const checksum = data.split('').reduce((sum, char) => sum +
char.charCodeAt(0), 0) % 97;

    return id.toString() + '-' + checksum;
}

// Decode a custom string ID (simplified - in practice this would require
the original charset)
function decodeCustomId(id) {
    // This is a placeholder since we can't fully decode without
additional information
    // In practice, the encoding would need to be reversible

    // Extract the checksum
    const parts = id.split('-');
    if (parts.length !== 2) {
        throw new Error('Invalid custom ID format');
    }

    return `[Original data (ID: ${parts[0]}, Checksum: ${parts[1]}])`;
}
```

```
// ===== 5.4 Transaction Operations =====
function setupTransactionOperations() {
    const drawingColor = document.getElementById('drawing-color');
    const drawingSize = document.getElementById('drawing-size');
    const undoBtn = document.getElementById('transaction-undo');
    const redoBtn = document.getElementById('transaction-redo');
    const clearBtn = document.getElementById('transaction-clear');
    const canvas = document.getElementById('drawing-canvas');
    const historyContainer = document.getElementById('transaction-history');
    const context = canvas.getContext('2d');

    // Transaction management
    let actionHistory = [];
    let redoStack = [];
    let isDrawing = false;
    let lastX = 0;
    let lastY = 0;

    // Set up canvas and initial state
    function initCanvas() {
        // Set canvas background
        context.fillStyle = 'white';
        context.fillRect(0, 0, canvas.width, canvas.height);

        // Add event listeners
        canvas.addEventListener('mousedown', startDrawing);
        canvas.addEventListener('mousemove', draw);
        canvas.addEventListener('mouseup', stopDrawing);
        canvas.addEventListener('mouseout', stopDrawing);

        // Touch support
        canvas.addEventListener('touchstart', handleTouchStart);
        canvas.addEventListener('touchmove', handleTouchMove);
        canvas.addEventListener('touchend', stopDrawing);

        updateButtons();
    }

    // Start drawing
    function startDrawing(e) {
        isDrawing = true;
        const pos = getPosition(e);
        lastX = pos.x;
        lastY = pos.y;

        // Start a new transaction
        saveInitialState();
    }

    // Draw on canvas
    function draw(e) {
        if (!isDrawing) return;
```

```
const pos = getPosition(e);
const currentX = pos.x;
const currentY = pos.y;

context.beginPath();
context.moveTo(lastX, lastY);
context.lineTo(currentX, currentY);
context.strokeStyle = drawingColor.value;
context.lineWidth = parseInt(drawingSize.value);
context.lineCap = 'round';
context.stroke();

lastX = currentX;
lastY = currentY;
}

// Stop drawing
function stopDrawing() {
    if (isDrawing) {
        isDrawing = false;
        saveFinalState();
    }
}

// Handle touch events
function handleTouchStart(e) {
    e.preventDefault();
    if (e.touches.length === 1) {
        const touch = e.touches[0];
        const mouseEvent = new MouseEvent('mousedown', {
            clientX: touch.clientX,
            clientY: touch.clientY
        });
        canvas.dispatchEvent(mouseEvent);
    }
}

function handleTouchMove(e) {
    e.preventDefault();
    if (e.touches.length === 1) {
        const touch = e.touches[0];
        const mouseEvent = new MouseEvent('mousemove', {
            clientX: touch.clientX,
            clientY: touch.clientY
        });
        canvas.dispatchEvent(mouseEvent);
    }
}

// Get mouse position relative to canvas
function getPosition(e) {
    const rect = canvas.getBoundingClientRect();
    return {
        x: e.clientX - rect.left,
```

```
        y: e.clientY - rect.top
    );
}

// Transaction handling
function saveInitialState() {
    // Capture the current canvas state
    const imageData = context.getImageData(0, 0, canvas.width,
    canvas.height);
    actionHistory.push({
        state: imageData,
        operation: 'Drawing started',
        color: drawingColor.value,
        size: drawingSize.value,
        timestamp: new Date().toISOString()
    });
}

// Clear redo stack when a new action is performed
redoStack = [];

updateHistory();
updateButtons();
}

function saveFinalState() {
    // Capture the final canvas state after drawing
    const imageData = context.getImageData(0, 0, canvas.width,
    canvas.height);
    actionHistory.push({
        state: imageData,
        operation: 'Drawing completed',
        color: drawingColor.value,
        size: drawingSize.value,
        timestamp: new Date().toISOString()
    });
}

updateHistory();
updateButtons();
}

// Undo the last action
function undo() {
    if (actionHistory.length <= 1) return;

    // Get the previous state
    const currentState = actionHistory.pop();
    redoStack.push(currentState);

    const previousState = actionHistory[actionHistory.length - 1];

    // Restore the previous state
    context.putImageData(previousState.state, 0, 0);

    updateHistory();
}
```

```
        updateButtons();
    }

    // Redo the last undone action
    function redo() {
        if (redoStack.length === 0) return;

        // Get the next state
        const nextState = redoStack.pop();
        actionHistory.push(nextState);

        // Restore the next state
        context.putImageData(nextState.state, 0, 0);

        updateHistory();
        updateButtons();
    }

    // Clear the canvas
    function clearCanvas() {
        // Save current state before clearing
        const imageData = context.getImageData(0, 0, canvas.width,
canvas.height);
        actionHistory.push({
            state: imageData,
            operation: 'Before clear',
            timestamp: new Date().toISOString()
        });

        // Clear the canvas
        context.fillStyle = 'white';
        context.fillRect(0, 0, canvas.width, canvas.height);

        // Save the cleared state
        const clearedImageData = context.getImageData(0, 0, canvas.width,
canvas.height);
        actionHistory.push({
            state: clearedImageData,
            operation: 'Canvas cleared',
            timestamp: new Date().toISOString()
        });

        // Clear redo stack
        redoStack = [];

        updateHistory();
        updateButtons();
    }

    // Update transaction history display
    function updateHistory() {
        historyContainer.innerHTML = '';

        // Display last 5 transactions
    }
}
```

```
const start = Math.max(0, actionHistory.length - 5);
for (let i = start; i < actionHistory.length; i++) {
    const transaction = actionHistory[i];
    const item = document.createElement('div');
    item.className = 'transaction-item';

    const time = new Date(transaction.timestamp).toLocaleTimeString();
    item.textContent = `${time}: ${transaction.operation}`;

    if (transaction.color) {
        const colorSwatch = document.createElement('span');
        colorSwatch.className = 'color-swatch';
        colorSwatch.style.backgroundColor = transaction.color;
        colorSwatch.style.display = 'inline-block';
        colorSwatch.style.width = '10px';
        colorSwatch.style.height = '10px';
        colorSwatch.style.marginLeft = '5px';

        item.appendChild(colorSwatch);
    }

    historyContainer.appendChild(item);
}
}

// Update button states
function updateButtons() {
    undoBtn.disabled = actionHistory.length <= 1;
    redoBtn.disabled = redoStack.length === 0;
}

// Initialize
initCanvas();

// Add event listeners
undoBtn.addEventListener('click', undo);
redoBtn.addEventListener('click', redo);
clearBtn.addEventListener('click', clearCanvas);
}

// ====== INITIALIZATION ======

/**
 * Initialize the application
 */
function init() {
    // Setup error modal close button
    const errorModal = document.getElementById('error-modal');
    const resultModal = document.getElementById('result-modal');
    const closeButton = document.querySelectorAll('.close-btn');

    closeButton.forEach(button => {
        button.addEventListener('click', function() {
            errorModal.style.display = 'none';
        });
    });
}
```

```
        resultModal.style.display = 'none';
    });
});

// Add to window event listeners in the init function
window.addEventListener('beforeunload', function() {
    // Explicitly clear all sensitive data structures
    virtualFS.root.children = {};
    database.deleteAll();
    memoryManager.clearAll();
    // Other data structures as needed
});

// Close modals when clicking outside
window.addEventListener('click', function(event) {
    if (event.target === errorModal) {
        errorModal.style.display = 'none';
    }
    if (event.target === resultModal) {
        resultModal.style.display = 'none';
    }
});

// Setup navigation
setupNavigation();

// Setup all operations
setupArrayOperations();
setupObjectOperations();
setupSetOperations();
setupTreeOperations();
setupRecursionOperations();
setupIterationOperations();
setupSearchOperations();
setupBinaryOperations();
setupFileOperations();
setupMemoryOperations();
setupProcessOperations();

// CRUD operations - Create returns an object with updateDatabaseDisplay
function
    // which is passed to the other CRUD operations
    const databaseDisplayUpdater = setupCreateOperations();
    setupReadOperations(databaseDisplayUpdater);
    setupUpdateOperations(databaseDisplayUpdater);
    setupDeleteOperations(databaseDisplayUpdater);

    setupSerializationOperations();
    setupCompressionOperations();
    setupEncodingOperations();
    setupTransactionOperations();

}
```

```
// Wait for DOM to be fully loaded before initializing
document.addEventListener('DOMContentLoaded', init);
})());
```

## [102] [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/CMakeLists.txt](#)

- **Bytes:** 2779
- **Type:** text

```
cmake_minimum_required(VERSION 3.28)
project(Linecraft LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 23)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)

add_executable(linecraft-cli
    src/linecraft_cli.cpp
)

if (MSVC)
    target_compile_options(linecraft-cli PRIVATE /W4 /permissive- /Zc:preprocessor
/utf-8)

# -----
# MSVC + Ninja can occasionally pick up x86 Windows/VC libraries even when the
# compiler is targeting x64 (environment ordering issues). That produces a
# wall of unresolved externals (operator new/delete, std::cout, filesystem,
# mainCRTStartup) and warnings like:
#   library machine type 'x86' conflicts with target machine type 'x64'
#
# Linecraft is intended to be "zero surprise" (CLIx5 philosophy): deterministic
# builds and predictable toolchain selection. So we defensively prepend the
# x64 library search paths based on the current VS/SDK environment variables.
#
option(LINECRAFT_FORCE_X64_LIBPATH "Prepend VC/Windows SDK x64 lib paths when
building 64-bit with MSVC" ON)

if (LINECRAFT_FORCE_X64_LIBPATH AND CMAKE_SIZEOF_VOID_P EQUAL 8)
    set(_vctools "$ENV{VCToolsInstallDir}")
    set(_winsdk  "$ENV{WindowsSdkDir}")
    set(_winsdkver "$ENV{WindowsSDKVersion}")

    # UCRT variables are sometimes separate
    set(_ucrtroot "$ENV{UniversalCRTSdkDir}")
    set(_ucrtver  "$ENV{UCRTVersion}")
    if (NOT _ucrtroot OR NOT _ucrtver)
        set(_ucrtroot "${_winsdk}")
        set(_ucrtver  "${_winsdkver}")
    endif()
```

```

# Normalize versions (strip slashes)
foreach(_v IN ITEMS _winsdkver _ucrtver)
    if (DEFINED ${_v})
        string(REGEX REPLACE "[/\\]+\$" "" ${_v} "${${_v}}")
    endif()
endforeach()

if (_vctools AND _winsdk AND _winsdkver)
    file(TO_CMAKE_PATH "${_vctools}/lib/x64" _vc_lib_x64)
    file(TO_CMAKE_PATH "${_winsdk}/Lib/${_winsdkver}/um/x64" _sdk_um_x64)
    if (_ucrroot AND _ucrtver)
        file(TO_CMAKE_PATH "${_ucrroot}/Lib/${_ucrtver}/ucrt/x64" _sdk_ucrt_x64)
    else()
        set(_sdk_ucrt_x64 "")
    endif()

    # Only add directories that exist (keeps cross setups clean).
    set(_lc_link_dirs "")
    foreach(_p IN ITEMS "${_vc_lib_x64}" "${_sdk_um_x64}" "${_sdk_ucrt_x64}")
        if (_p AND EXISTS "${_p}")
            list(APPEND _lc_link_dirs "${_p}")
        endif()
    endforeach()

    if (_lc_link_dirs)
        message(STATUS "Linecraft: Prepending x64 library paths: ${_lc_link_dirs}")
        target_link_directories(linecraft-cli BEFORE PRIVATE ${_lc_link_dirs})
    else()
        message(WARNING "Linecraft: Could not resolve x64 library directories. If
you see x86/x64 link conflicts, run an x64 VS Developer shell.")
    endif()
    endif()
endif()

else()
    target_compile_options(linecraft-cli PRIVATE -Wall -Wextra -Wpedantic)
endif()

```

### [103] Linecraft\_cpp\_with\_emit\_rootfix2/CMakePresets.json

- **Bytes:** 1725
- **Type:** text

```
{
  "version": 6,
  "cmakeMinimumRequired": { "major": 3, "minor": 28, "patch": 0 },
  "configurePresets": [
    {
      "name": "windows-msvc-debug",
      "displayName": "Windows MSVC Debug (Ninja)",

```

```

"generator": "Ninja",
"binaryDir": "${sourceDir}/out/build/${presetName}",
"cacheVariables": { "CMAKE_BUILD_TYPE": "Debug" },
"condition": { "type": "equals", "lhs": "${hostSystemName}", "rhs":
"Windows" }
},
{
  "name": "windows-msvc-release",
  "displayName": "Windows MSVC Release (Ninja)",
  "generator": "Ninja",
  "binaryDir": "${sourceDir}/out/build/${presetName}",
  "cacheVariables": { "CMAKE_BUILD_TYPE": "Release" },
  "condition": { "type": "equals", "lhs": "${hostSystemName}", "rhs":
"Windows" }
},
{
  "name": "linux-debug",
  "displayName": "Linux Debug (Ninja)",
  "generator": "Ninja",
  "binaryDir": "${sourceDir}/out/build/${presetName}",
  "cacheVariables": { "CMAKE_BUILD_TYPE": "Debug" },
  "condition": { "type": "equals", "lhs": "${hostSystemName}", "rhs": "Linux"
}
},
{
  "name": "macos-debug",
  "displayName": "macOS Debug (Ninja)",
  "generator": "Ninja",
  "binaryDir": "${sourceDir}/out/build/${presetName}",
  "cacheVariables": { "CMAKE_BUILD_TYPE": "Debug" },
  "condition": { "type": "equals", "lhs": "${hostSystemName}", "rhs": "Darwin"
}
}
],
"buildPresets": [
  { "name": "windows-msvc-debug", "configurePreset": "windows-msvc-debug" },
  { "name": "windows-msvc-release", "configurePreset": "windows-msvc-release" },
  { "name": "linux-debug", "configurePreset": "linux-debug" },
  { "name": "macos-debug", "configurePreset": "macos-debug" }
]
}

```

[104] [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/files/x00001.txt](#)

- **Bytes:** 46
- **Type:** text

```
x00001 (demo context){
 0001   print("hello")
```

```
}
```

## [105] Linecraft\_cpp\_with\_emit\_rootfix2/files/x00002.txt

- **Bytes:** 143
- **Type:** text

```
x00002 (emit demo){  
01  
0001     #include <iostream>  
0002     int main(){  
0003         std::cout << "Hello from Linecraft\\n";  
0004         return 0;  
0005     }  
}
```

## [106] Linecraft\_cpp\_with\_emit\_rootfix2/plugins/echo/echo\_plugin.py

- **Bytes:** 953
- **Type:** text

```
import json, os, sys  
  
def main():  
    if len(sys.argv) < 3:  
        print("usage: echo_plugin.py <input.json> <outdir>", file=sys.stderr)  
        return 2  
    input_path = sys.argv[1]  
    outdir = sys.argv[2]  
    os.makedirs(outdir, exist_ok=True)  
  
    with open(input_path, "r", encoding="utf-8") as f:  
        inp = json.load(f)  
  
        code_file = inp.get("code_file", "")  
        try:  
            with open(code_file, "r", encoding="utf-8") as f:  
                code = f.read()  
        except Exception as e:  
            code = f"[cannot read code_file: {e}]"  
  
        out = {  
            "ok": True,  
            "plugin": "echo",  
            "echo": code,  
            "stdin": inp.get("stdin", {}),  
            "bank": inp.get("bank"),
```

```

        "reg": inp.get("reg"),
        "addr": inp.get("addr"),
    }

    with open(os.path.join(outdir, "output.json"), "w", encoding="utf-8") as f:
        json.dump(out, f, indent=2)

    return 0

if __name__ == "__main__":
    raise SystemExit(main())

```

## [107] Linecraft\_cpp\_with\_emit\_rootfix2/plugins/echo/plugin.json

- **Bytes:** 72
- **Type:** text

```
{
    "name": "echo",
    "entry_win": "run.bat",
    "entry_lin": "run.sh"
}
```

## [108] Linecraft\_cpp\_with\_emit\_rootfix2/plugins/emit/emit\_plugin.py

- **Bytes:** 7734
- **Type:** text

```

import json, os, sys
from pathlib import Path

def _die(msg: str, code: int = 2):
    print(msg, file=sys.stderr)
    return code

def _read_json(p: Path):
    with p.open("r", encoding="utf-8") as f:
        return json.load(f)

def _write_json(p: Path, obj):
    p.parent.mkdir(parents=True, exist_ok=True)
    with p.open("w", encoding="utf-8") as f:
        json.dump(obj, f, indent=2)

def _safe_resolve_under(root: Path, rel: str, allow_outside: bool) -> Path:
    rel = rel.replace("\\\\", "/")
    if not allow_outside:

```

```
parts = [p for p in rel.split("/") if p not in ("", ".")]
if any(p == ".." for p in parts):
    raise ValueError("Path traversal '..' is not allowed: " + rel)
target = (root / Path(*parts)).resolve()
if root not in target.parents and target != root:
    raise ValueError("Output path escapes workspace root: " + str(target))
return target
return Path(rel).expanduser().resolve() if os.path.isabs(rel) else (root /
rel).resolve()

def _parse_context_values(context_path: Path):
    regs = {}
    current_reg = None
    with context_path.open("r", encoding="utf-8", errors="replace") as f:
        for raw_line in f:
            line = raw_line.rstrip("\r\n")
            t = line.strip()
            if not t:
                continue
            if t == "}":
                break
            if (not line.startswith((" ", "\t"))) and ("(" not in line) and ("{" not in line):
                current_reg = t
                regs.setdefault(current_reg, {})
                continue
            if line.startswith((" ", "\t")):
                s = line.lstrip(" \t")
                if "\t" in s:
                    addr, val = s.split("\t", 1)
                else:
                    parts = s.split(maxsplit=1)
                    addr = parts[0]
                    val = parts[1] if len(parts) > 1 else ""
                addr = addr.strip()
                if current_reg is None:
                    current_reg = "01"
                regs.setdefault(current_reg, {})
                regs[current_reg][addr] = val
    return regs

def _collect_from_spec(regs_map, spec, default_reg Tok):
    reg = str(spec.get("reg") or default_reg Tok)
    reg_map = regs_map.get(reg, {})
    out = []

    if "addrs" in spec:
        for a in spec["addrs"]:
            a = str(a)
            out.append(reg_map.get(a, ""))
    return out

    if "range" in spec:
        r = spec["range"]
```

```
a_from = str(r.get("from"))
a_to = str(r.get("to"))
keys = sorted(reg_map.keys())
for k in keys:
    if a_from <= k <= a_to:
        out.append(reg_map[k])
return out

if "addr" in spec:
    a = str(spec["addr"])
    out.append(reg_map.get(a, ""))
    return out

return out

def main():
    if len(sys.argv) < 3:
        return _die("usage: emit_plugin.py <input.json> <outdir>")

    input_path = Path(sys.argv[1])
    outdir = Path(sys.argv[2])
    outdir.mkdir(parents=True, exist_ok=True)

    inp = _read_json(input_path)

    cfg = inp.get("cfg", {})
    prefix = str(cfg.get("prefix", "x"))[:1]
    default_reg = cfg.get("defaultReg", 1)
    if isinstance(default_reg, int):
        default_reg_tok = f"{default_reg:02d}"
    else:
        default_reg_tok = str(default_reg)

    context_file = inp.get("context_file", "")
    if not context_file:
        bank = inp.get("bank", "")
        if not bank:
            return _die("missing bank in input.json")
        bank = str(bank)
        if not bank.startswith(prefix):
            bank = prefix + bank
        context_file = str(Path("files") / f"{bank}.txt")

    context_path = Path(context_file)
    if not context_path.exists():
        return _die(f"context file missing: {context_path}")

    regs_map = _parse_context_values(context_path)

    stdin = inp.get("stdin", {})
    if not isinstance(stdin, dict):
        return _die("stdin must be a JSON object")

    workspace_root = Path.cwd().resolve()
```

```
allow_outside = bool(stdin.get("allow_outside_workspace", False))

project_root = str(stdin.get("project_root", "files/out/project"))
try:
    project_abs = _safe_resolve_under(workspace_root, project_root,
allow_outside)
except Exception as e:
    return _die(f"bad project_root: {e}")

created_dirs = []
created_files = []

for d in (stdin.get("mkdir", []) or []):
    try:
        p = _safe_resolve_under(project_abs, str(d), allow_outside=True)
        p.mkdir(parents=True, exist_ok=True)
        created_dirs.append(str(p))
    except Exception as e:
        return _die(f"mkdir failed for '{d}': {e}")

overwrite = bool(stdin.get("allow_overwrite", False))
files = stdin.get("files", []) or []
if not isinstance(files, list):
    return _die("stdin.files must be a list")

for fdesc in files:
    if not isinstance(fdesc, dict):
        return _die("each files[] entry must be an object")

    rel_path = fdesc.get("path")
    if not rel_path:
        return _die("files[].path is required")

    try:
        file_abs = _safe_resolve_under(project_abs, str(rel_path),
allow_outside=True)
    except Exception as e:
        return _die(f"bad file path '{rel_path}': {e}")

    file_abs.parent.mkdir(parents=True, exist_ok=True)

    mode = fdesc.get("mode", "overwrite") # overwrite | append
    if file_abs.exists() and not overwrite and mode == "overwrite":
        created_files.append({"path": str(file_abs), "status": "skipped_exists"})
        continue

    joiner = fdesc.get("join", "\n")
    ensure_nl = bool(fdesc.get("ensure_final_newline", True))

    content = ""
    if "text" in fdesc:
        content = str(fdesc.get("text", ""))
    elif "from" in fdesc:
```

```
parts = []
from_list = fdesc.get("from", [])
if not isinstance(from_list, list):
    return _die("files[].from must be a list")
for spec in from_list:
    if isinstance(spec, str):
        if ":" in spec:
            r, a = spec.split(":", 1)
            spec = {"reg": r, "addr": a}
        else:
            spec = {"addr": spec}
    vals = _collect_from_spec(regs_map, spec, default_reg_tok)
    parts.extend(vals)
content = joiner.join(parts)
elif "range" in fdesc:
    vals = _collect_from_spec(regs_map, {"range": fdesc["range"], "reg": fdesc.get("reg")}, default_reg_tok)
    content = joiner.join(vals)
else:
    return _die(f"files[].text or files[].from or files[].range required for {rel_path}")

if ensure_nl and not content.endswith("\n"):
    content += "\n"

try:
    if mode == "append":
        with file_abs.open("a", encoding="utf-8", newline="\n") as out:
            out.write(content)
    else:
        with file_abs.open("w", encoding="utf-8", newline="\n") as out:
            out.write(content)
except Exception as e:
    return _die(f"write failed for '{rel_path}': {e}")

created_files.append({"path": str(file_abs), "status": "written", "bytes": len(content.encode("utf-8"))})

report = {
    "ok": True,
    "plugin": "emit",
    "workspace_root": str(workspace_root),
    "project_root": str(project_abs),
    "created_dirs": created_dirs,
    "files": created_files
}
_write_json(outdir / "output.json", report)
return 0

if __name__ == "__main__":
    raise SystemExit(main())
```

## [109] [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/plugins/emit/plugin.json](#)

- **Bytes:** 72
- **Type:** text

```
{
  "name": "emit",
  "entry_win": "run.bat",
  "entry_lin": "run.sh"
}
```

## [110] [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/README.md](#)

- **Bytes:** 3455
- **Type:** text

### # Linecraft (C++ / Visual Studio 2026 Insiders)

This repo is a **\*\*non-AI\*\*** baseline implementation of the clix-style context system captured in `doc.txt`.

What you get:

- Context text format parsing/serialization (bank/register/address)
- A compatible REPL command set (`:open`, `:ins`, `:insr`, `:w`, `:resolve`, `:export`, ...)
- **\*\*Optional\*\* \*offline\* filesystem plugins` (:plugin\_run) - **\*\*no AI dependency\*\*****

### ## Build (Windows / VS 2026 Insiders)

#### ### Visual Studio (recommended)

- **\*\*File → Open → Folder...\*\*** (open this repo)
- Choose preset: `windows-msvc-debug`
- Build target: `linecraft-cli`

#### ### Command line

From “Developer PowerShell for VS”:

```
```powershell
cmake --preset windows-msvc-debug
cmake --build --preset windows-msvc-debug
```
```

Binary: `out/build/windows-msvc-debug/linecraft-cli.exe`

#### ## Run

Run from repo root (so `files/` and `plugins/` are found):

```
```powershell
out\build\windows-msvc-debug\linecraft-cli.exe
```

```
```
```

Try:

```
```text
:open x00001
:ins 0001 print("hello")
:w
:resolve
:export
:plugins
:plugin_run echo 01 0001 {"note":"demo"}
:q
```

```

Outputs:

- `files/out/x00001.resolved.txt`
- `files/out/x00001.json`
- `files/out/plugins/x00001/r01a0001/echo/output.json`

### [## Design note: removing AI plugin components](#)

The prototype design in `doc.txt` mentions “AI plugin” ideas (Codex/Gemini) as a computation oracle.

Linecraft removes that requirement:

- Core editing/resolve/export works with **\*\*zero plugins\*\***.
- Plugins are treated as **\*\*local programs\*\*** invoked via a manifest (`plugins/<name>/plugin.json`).
- Nothing in the host assumes network, LLM APIs, or AI-specific IO.

### [## Default plugins shipped](#)

#### [### 1\) `echo`](#)

Minimal demo plugin (offline) – returns the resolved cell content in `output.json`.

#### [### 2\) `emit`](#)

Offline **\*\*codebase emitter\*\*** – creates directories recursively and writes one/multiple files by joining address values as multi-line text. Supports any file extension (path determines extension).

#### **\*\*Emit a file from multiple addresses\*\***

```
```text
:open x00002
:plugin_run emit 01 0001 {"project_root":"files/out/projects/hello","mkdir": ["src"], "allow_overwrite":true, "files": [ {"path": "src/main.cpp", "from": [{"reg": "01", "addrs": ["0001", "0002", "0003", "0004", "0005"]}], "join": "\n"} ] }
```

```

```
```
**Create a multi-folder project scaffold**
```text
:plugin_run emit 01 0001 {"project_root":"files/out/projects/MyLib","mkdir": ["include/mylib","src","tests"], "allow_overwrite":true, "files": [ {"path":"include/mylib/mylib.hpp","text": "#pragma once\n\n// ..."}, {"path":"src/mylib.cpp","text": "#include \"mylib/mylib.hpp\"\n"}, {"path":"CMakeLists.txt","text": "cmake_minimum_required(VERSION 3.28)\nproject(MyLib LANGUAGES CXX)\n"} ]
}
```

```

### Notes

- By default, output is restricted to **\*\*inside the current workspace root\*\*** (the folder you run Linecraft from).
- Use `allow\_outside\_workspace": true` only if you intentionally want to write outside the workspace.

### ## Windows troubleshooting (MSVC + Ninja)

If you see a linker wall like `LNK2019` plus warnings:

```
`library machine type 'x86' conflicts with target machine type 'x64'`  

it means your link search path is resolving **x86** Windows/VC libraries while your build is targeting **x64**.  

This repo defensively prepends the expected x64 lib paths (see `LINECRAFT_FORCE_X64_LIBPATH` in `CMakeLists.txt`), but you should also ensure you are using an **x64** Visual Studio Developer PowerShell / Command Prompt.
```

## [111] [Linecraft\\_cpp\\_with\\_emit\\_rootfix2/src/linecraft\\_cli.cpp](#)

- **Bytes:** 37130
- **Type:** text

```
#include <algorithm>
#include <cctype>
#include <cstdio>
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <map>
#include <optional>
#include <regex>
```

```
#include <sstream>
#include <string>
#include <unordered_set>
#include <vector>
#include <array>

#ifndef _WIN32
# ifndef WIN32_LEAN_AND_MEAN
#   define WIN32_LEAN_AND_MEAN 1
# endif
# ifndef NOMINMAX
#   define NOMINMAX 1
# endif
# include <windows.h>
#endif

#if defined(__APPLE__)
# include <mach-o/dyld.h>
#endif

#if !defined(_WIN32)
# include <unistd.h>
#endif

namespace fs = std::filesystem;

static void init_console_utf8() {
#ifndef _WIN32
    // Best-effort: ensure UTF-8 output so box drawing / em-dash render correctly
    // even when the host console code page defaults to CP437/850.
    ::SetConsoleOutputCP(CP_UTF8);
    ::SetConsoleCP(CP_UTF8);

    // Optional: allow ANSI/VT sequences when available (harmless if not
    // supported).
    HANDLE hOut = ::GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut != INVALID_HANDLE_VALUE) {
        DWORD mode = 0;
        if (::GetConsoleMode(hOut, &mode)) {
            mode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
            ::SetConsoleMode(hOut, mode);
        }
    }
#endif
}

//=====
// Runtime path discovery
=====

struct RuntimePaths {
    fs::path root;      // Project root (contains plugins/ and files/)
    fs::path files;     // root/files
```

```
fs::path out;      // root/files/out
fs::path plugins; // root/plugins
};

static fs::path get_executable_pathFallback(const char* argv0) {
    // argv0 is not reliable on Windows but can help on other platforms.
    if (!argv0 || !*argv0) return {};
    try {
        fs::path p = fs::path(argv0);
        if (p.is_relative()) p = fs::current_path() / p;
        return fs::weakly_canonical(p);
    } catch (...) {
        return {};
    }
}

static fs::path get_executable_path(const char* argv0) {
#ifndef _WIN32
    std::wstring buf(MAX_PATH, L'\0');
    DWORD len = ::GetModuleFileNameW(nullptr, buf.data(), static_cast<DWORD>(buf.size()));
    if (len > 0) {
        buf.resize(len);
        return fs::path(buf);
    }
    return get_executable_pathFallback(argv0);
#elif defined(__APPLE__)
    uint32_t size = 0;
    (void)_NSGetExecutablePath(nullptr, &size);
    if (size > 0) {
        std::string tmp(size, '\0');
        if (_NSGetExecutablePath(tmp.data(), &size) == 0) {
            return fs::weakly_canonical(fs::path(tmp.c_str()));
        }
    }
    return get_executable_pathFallback(argv0);
#else
    std::array<char, 4096> buf{};
    ssize_t n = ::readlink("/proc/self/exe", buf.data(), buf.size() - 1);
    if (n > 0) {
        buf[static_cast<size_t>(n)] = '\0';
        return fs::weakly_canonical(fs::path(buf.data()));
    }
    return get_executable_pathFallback(argv0);
#endif
}

static bool looks_like_root(const fs::path& p) {
    // Minimal heuristic: a plugins directory exists.
    // We don't require files/ because the CLI can create it.
    return fs::exists(p / "plugins") && fs::is_directory(p / "plugins");
}

static fs::path detect_root(const char* argv0) {
```

```
// Search order: CWD then exe-dir, then walk up a few parents.
std::vector<fs::path> starts;
try { starts.push_back(fs::current_path()); } catch (...) {}

fs::path exe = get_executable_path(argv0);
if (!exe.empty()) starts.push_back(exe.parent_path());

for (const auto& start : starts) {
    fs::path cur = start;
    for (int i = 0; i < 8; ++i) {
        if (cur.empty()) break;
        if (looks_like_root(cur)) return cur;
        // Secondary hint: CMake project root.
        if (fs::exists(cur / "CMakeLists.txt") && fs::exists(cur / "plugins"))
return cur;
        fs::path parent = cur.parent_path();
        if (parent == cur) break;
        cur = parent;
    }
}
// Fallback to current directory.
try { return fs::current_path(); } catch (...) { return {}; }

static RuntimePaths make_runtime_paths(const char* argv0) {
    RuntimePaths rp;
    rp.root = detect_root(argv0);
    if (rp.root.empty()) {
        try { rp.root = fs::current_path(); } catch (...) { rp.root =
fs::path("."); }
    }
    rp.files = rp.root / "files";
    rp.out = rp.files / "out";
    rp.plugins = rp.root / "plugins";
    return rp;
}

// -----
// Config (mirrors the clix examples in doc.txt; kept small + deterministic)
// -----
struct Config {
    char prefix = 'x';
    int base = 10;
    int widthBank = 5;
    int widthReg = 2;
    int widthAddr = 4;
    long long defaultReg = 1; // "01" (doc.txt help: :ins targets register 1)
    RuntimePaths paths;
};

static std::string trim(std::string s) {
    auto ns = [] (int ch){ return !std::ispace(ch); };
    s.erase(s.begin(), std::find_if(s.begin(), s.end(), ns));
}
```

```
s.erase(std::find_if(s.rbegin(), s.rend(), ns).base(), s.end());
return s;
}

static int digit_value(char c){
    if (c>='0' && c<='9') return c-'0';
    if (c>='A' && c<='Z') return 10+(c-'A');
    if (c>='a' && c<='z') return 10+(c-'a');
    return -1;
}

static bool parse_int_base(const std::string& s, int base, long long& out){
    if (s.empty()) return false;
    long long v=0;
    for (char c : s) {
        int d = digit_value(c);
        if (d < 0 || d >= base) return false;
        v = v*base + d;
        if (v < 0) return false;
    }
    out = v;
    return true;
}

static std::string to_base_n(long long val, int base, int width){
    if (base < 2 || base > 36) base = 10;
    if (val == 0) return std::string(std::max(1, width), '0');
    bool neg = val < 0; if (neg) val = -val;
    std::string s;
    while (val > 0) {
        int d = int(val % base);
        s.push_back(d < 10 ? char('0' + d) : char('a' + (d - 10)));
        val /= base;
    }
    if (neg) s.push_back('-');
    std::reverse(s.begin(), s.end());
    if (width > 0 && (int)s.size() < width) s = std::string(width - (int)s.size(), '0') + s;
    return s;
}

static std::string json_escape(const std::string& s){
    std::string o; o.reserve(s.size()+8);
    for (unsigned char c : s) {
        switch (c) {
            case '\\': o += "\\\\"; break;
            case '\"': o += "\\\""; break;
            case '\\b': o += "\\b"; break;
            case '\\f': o += "\\f"; break;
            case '\\n': o += "\\n"; break;
            case '\\r': o += "\\r"; break;
            case '\\t': o += "\\t"; break;
            default:
                if (c < 0x20) { char buf[7]; std::snprintf(buf, sizeof(buf),
```

```
"\\u%04X", c); o += buf; }
        else o += char(c);
    }
}
return o;
}

// -----
// Model
// -----
struct Bank {
    long long id = 0;
    std::string title;
    std::map<long long, std::map<long long, std::string>> regs; // reg -> addr ->
value
};

struct Workspace {
    std::map<long long, Bank> banks;
    std::map<long long, fs::path> files;
};

static fs::path ctx_path(const Config& cfg, long long bank){
    return cfg.paths.files / (std::string(1, cfg.prefix) + to_base_n(bank,
cfg.base, cfg.widthBank) + ".txt");
}
static fs::path out_resolved_path(const Config& cfg, long long bank){
    return cfg.paths.out / (std::string(1, cfg.prefix) + to_base_n(bank, cfg.base,
cfg.widthBank) + ".resolved.txt");
}
static fs::path out_json_path(const Config& cfg, long long bank){
    return cfg.paths.out / (std::string(1, cfg.prefix) + to_base_n(bank, cfg.base,
cfg.widthBank) + ".json");
}

static bool read_all(const fs::path& p, std::string& out){
    std::ifstream in(p, std::ios::binary);
    if (!in) return false;
    out.assign((std::istreambuf_iterator<char>(in)), {});
    return true;
}
static bool write_all(const fs::path& p, const std::string& s){
    fs::create_directories(p.parent_path());
    auto tmp = p; tmp += ".tmp";
    {
        std::ofstream out(tmp, std::ios::binary | std::ios::trunc);
        if (!out) return false;
        out.write(s.data(), (std::streamsize)s.size());
        if (!out) return false;
    }
    std::error_code ec;
    fs::rename(tmp, p, ec);
    if (ec) {
        fs::copy_file(tmp, p, fs::copy_options::overwrite_existing, ec);
    }
}
```

```
        fs::remove(tmp);
        if (ec) return false;
    }
    return true;
}

// -----
// Context parsing/writing (compatible with doc.txt examples)
// - Header: x0001\t(title){
// - Optional register lines (00, 02, ...)
// - Address lines indented: \t0001\tvalue
// - If no register line appears, defaultReg is used.
// -----

static bool parse_bank_text(const std::string& text, const Config& cfg, Bank& out,
                           std::string& err){
    std::string content = text;
    if (content.size() >= 3 && (unsigned char)content[0]==0xEF && (unsigned
char)content[1]==0xBB && (unsigned char)content[2]==0xBF)
        content.erase(0,3);

    std::vector<std::string> lines;
    {
        std::istringstream is(content);
        std::string line;
        while (std::getline(is, line)) lines.push_back(line);
    }
    if (lines.empty()) { err = "empty"; return false; }

    size_t i=0;
    while (i<lines.size() && trim(lines[i]).empty()) i++;
    if (i==lines.size()) { err = "no header"; return false; }

    std::string header = trim(lines[i]);
    size_t j=i+1;
    while (header.find('{')==std::string::npos && j<lines.size()) {
        header += " " + trim(lines[j]);
        j++;
    }
    auto lp = header.find('(');
    auto rp = header.rfind(')');
    if (lp==std::string::npos || rp==std::string::npos || rp<lp) { err = "bad
header"; return false; }

    std::string left = trim(header.substr(0, lp));
    std::string title = trim(header.substr(lp+1, rp-lp-1));
    if (!left.empty() && left[0]==cfg.prefix) left = left.substr(1);

    long long bankId=0;
    if (!parse_int_base(left, cfg.base, bankId)) { err = "bad bank id"; return
false; }

    out = {};
    out.id = bankId;
    out.title = title;
```

```
size_t body = i;
while (body<lines.size() && lines[body].find('{')==std::string::npos) body++;
if (body==lines.size()) { err = "missing {"; return false; }
body++;

long long curReg = cfg.defaultReg;

for (size_t k=body; k<lines.size(); ++k) {
    std::string s = lines[k];
    if (s.find('}')!=std::string::npos) break;
    if (trim(s).empty()) continue;

    if (!s.empty() && s[0] != '\t' && s[0] != ' ') {
        long long reg=0;
        if (!parse_int_base(trim(s), cfg.base, reg)) { err = "bad register
line"; return false; }
        curReg = reg;
        continue;
    }

    while (!s.empty() && (s[0]=='\t' || s[0]==' ')) s.erase(s.begin());
    size_t sep = s.find('\t');
    if (sep==std::string::npos) sep = s.find(' ');
    std::string addrTok, val;
    if (sep==std::string::npos) { addrTok = trim(s); val = ""; }
    else { addrTok = trim(s.substr(0, sep)); val = s.substr(sep+1); }

    long long addr=0;
    if (!parse_int_base(addrTok, cfg.base, addr)) { err = "bad addr"; return
false; }
    out.regs[curReg][addr] = val;
}

return true;
}

static std::string write_bank_text(const Bank& b, const Config& cfg){
    std::ostringstream os;
    std::string bankStr = std::string(1, cfg.prefix) + to_base_n(b.id, cfg.base,
cfg.widthBank);
    os << bankStr << "\t(" << b.title << "){\n";

    bool multi = (b.regs.size()>1) || (b.regs.size()==1 && b.regs.begin()->first
!= cfg.defaultReg);
    if (!multi) {
        auto it = b.regs.find(cfg.defaultReg);
        if (it != b.regs.end()) {
            for (auto& [addr, val] : it->second)
                os << "\t" << to_base_n(addr, cfg.base, cfg.widthAddr) << "\t" <<
val << "\n";
        }
    } else {
        for (auto& [reg, addrs] : b.regs) {
```

```
        os << to_base_n(reg, cfg.base, cfg.widthReg) << "\n";
        for (auto& [addr, val] : addrs)
            os << "\t" << to_base_n(addr, cfg.base, cfg.widthAddr) << "\t" <<
    val << "\n";
}
}

os << "}\n";
return os.str();
}

static bool ensure_loaded(const Config& cfg, Workspace& ws, long long bank,
std::string& err){
    if (ws.banks.count(bank)) return true;
    auto p = ctx_path(cfg, bank);
    if (!fs::exists(p)) { err = "missing: " + p.string(); return false; }
    std::string text;
    if (!read_all(p, text)) { err = "cannot read: " + p.string(); return false; }
    Bank b;
    if (!parse_bank_text(text, cfg, b, err)) return false;
    ws.banks[bank] = std::move(b);
    ws.files[bank] = p;
    return true;
}

// -----
// Resolver (supports references described in doc.txt):
//   r<reg>.<addr>
//   x<bank>.<reg>.<addr>
//   <bank>.<reg>.<addr>
//   x<bank>.<addr> (defaults reg=defaultReg)
//   @file(path)
// Circular refs protected.
// -----
struct Resolver {
    const Config& cfg;
    Workspace& ws;

    bool get_value(long long bank, long long reg, long long addr, std::string&
out){
        std::string err; (void)ensure_loaded(cfg, ws, bank, err);
        auto itB = ws.banks.find(bank);
        if (itB==ws.banks.end()) return false;
        auto itR = itB->second.regs.find(reg);
        if (itR==itB->second.regs.end()) return false;
        auto itA = itR->second.find(addr);
        if (itA==itR->second.end()) return false;
        out = itA->second;
        return true;
    }

    std::string resolve_string(const std::string& in, long long currentBank,
std::unordered_set<std::string> visited){
        std::string s = in;
```

```
// @file(path)
{
    static const std::regex re(R"(@file\(((^)]+)\))");
    std::smatch m;
    std::string out; out.reserve(s.size());
    auto it=s.cbegin(), end=s.cend();
    while (std::regex_search(it, end, m, re)) {
        out.append(it, m[0].first);
        auto fname = trim(m[1].str());
        std::string txt;
        if (!read_all(fs::path(fname), txt)) out += "[Missing file: " +
fname + "]";
        else out += txt;
        it = m[0].second;
    }
    out.append(it, end);
    s.swap(out);
}

auto resolve_ref = [&](long long b, long long r, long long a, const
std::string& token)->std::string{
    std::string key = std::to_string(b) + "." + std::to_string(r) + "." +
std::to_string(a);
    if (visited.count(key)) return "[Circular Ref: " + token + "]";
    std::string v;
    if (!get_value(b, r, a, v)) return "[Missing " + token + "]";
    visited.insert(key);
    return resolve_string(v, b, visited);
};

// r<reg>.<addr>
{
    static const std::regex re(R"(r([0-9A-Za-z]+)\.([0-9A-Za-z]+))");
    std::smatch m;
    std::string out; out.reserve(s.size());
    auto it=s.cbegin(), end=s.cend();
    while (std::regex_search(it, end, m, re)) {
        out.append(it, m[0].first);
        long long r=0,a=0;
        if (!parse_int_base(m[1].str(), cfg.base, r) ||
!parse_int_base(m[2].str(), cfg.base, a)) out += "[BadRef " + m[0].str() + "]";
        else out += resolve_ref(currentBank, r, a, m[0].str());
        it = m[0].second;
    }
    out.append(it, end);
    s.swap(out);
}

// x<bank>.<reg>.<addr>
{
    const std::regex re(std::string(1,cfg.prefix) + R"(([0-9A-Za-z]+)\.
([0-9A-Za-z]+)\.([0-9A-Za-z]+))");
    std::smatch m;
```

```
    std::string out; out.reserve(s.size());
    auto it=s.cbegin(), end=s.cend();
    while (std::regex_search(it, end, m, re)) {
        out.append(it, m[0].first);
        long long b=0,r=0,a=0;
        if (!parse_int_base(m[1].str(), cfg.base, b) ||
!parse_int_base(m[2].str(), cfg.base, r) || !parse_int_base(m[3].str(), cfg.base,
a)) out += "[BadRef " + m[0].str() + "]";
        else out += resolve_ref(b, r, a, m[0].str());
        it = m[0].second;
    }
    out.append(it, end);
    s.swap(out);
}

// <bank>.<reg>.<addr>
{
    static const std::regex re(R"(\b([0-9A-Za-z]+)\.([0-9A-Za-z]+)\.([0-
9A-Za-z]+)\b)");
    std::smatch m;
    std::string out; out.reserve(s.size());
    auto it=s.cbegin(), end=s.cend();
    while (std::regex_search(it, end, m, re)) {
        out.append(it, m[0].first);
        long long b=0,r=0,a=0;
        if (!parse_int_base(m[1].str(), cfg.base, b) ||
!parse_int_base(m[2].str(), cfg.base, r) || !parse_int_base(m[3].str(), cfg.base,
a)) out += "[BadRef " + m[0].str() + "]";
        else out += resolve_ref(b, r, a, m[0].str());
        it = m[0].second;
    }
    out.append(it, end);
    s.swap(out);
}

// x<bank>.<addr> (defaults reg=defaultReg)
{
    const std::regex re(std::string(1,cfg.prefix) + R"(([0-9A-Za-z]+)\.([0-
9A-Za-z]+))");
    std::smatch m;
    std::string out; out.reserve(s.size());
    auto it=s.cbegin(), end=s.cend();
    while (std::regex_search(it, end, m, re)) {
        auto tok = m[0].str();
        if (std::count(tok.begin(), tok.end(), '.') != 1) { out.append(it,
m[0].second); it = m[0].second; continue; }
        out.append(it, m[0].first);
        long long b=0,a=0;
        if (!parse_int_base(m[1].str(), cfg.base, b) ||
!parse_int_base(m[2].str(), cfg.base, a)) out += "[BadRef " + tok + "]";
        else out += resolve_ref(b, cfg.defaultReg, a, tok);
        it = m[0].second;
    }
    out.append(it, end);
}
```

```
        s.swap(out);
    }

    return s;
}

};

static std::string resolve_bank_to_text(const Config& cfg, Workspace& ws, long long bank){
    std::string err; (void)ensure_loaded(cfg, ws, bank, err);
    auto itB = ws.banks.find(bank);
    if (itB==ws.banks.end()) return {};
    Resolver R{cfg, ws};

    const Bank& b = itB->second;
    std::ostringstream os;
    std::string bankStr = std::string(1, cfg.prefix) + to_base_n(b.id, cfg.base, cfg.widthBank);
    os << bankStr << "\t(" << b.title << "){\n";

    bool multi = (b.regs.size()>1) || (b.regs.size()==1 && b.regs.begin()->first != cfg.defaultReg);
    if (!multi) {
        auto it = b.regs.find(cfg.defaultReg);
        if (it != b.regs.end()) {
            for (auto& [addr, raw] : it->second) {
                auto resolved = R.resolve_string(raw, bank, {});
                os << "\t" << to_base_n(addr, cfg.base, cfg.widthAddr) << "\t" << resolved << "\n";
            }
        }
    } else {
        for (auto& [reg, addrs] : b.regs) {
            os << to_base_n(reg, cfg.base, cfg.widthReg) << "\n";
            for (auto& [addr, raw] : addrs) {
                auto resolved = R.resolve_string(raw, bank, {});
                os << "\t" << to_base_n(addr, cfg.base, cfg.widthAddr) << "\t" << resolved << "\n";
            }
        }
    }
    os << "}\n";
    return os.str();
}

static std::string export_bank_to_json(const Config& cfg, Workspace& ws, long long bank){
    std::string err; (void)ensure_loaded(cfg, ws, bank, err);
    auto itB = ws.banks.find(bank);
    if (itB==ws.banks.end()) return "{}\n";

    const Bank& b = itB->second;
    std::string bankStr = std::string(1, cfg.prefix) + to_base_n(b.id, cfg.base, cfg.widthBank);
```

```

    std::ostringstream os;
    os << "{\n";
    os << "  \"bank\": \"\" << json_escape(bankStr) << "\",\n";
    os << "  \"title\": \"\" << json_escape(b.title) << "\",\n";
    os << "  \"registers\": [\n";

    bool firstReg=true;
    for (auto& [reg, addrs] : b.regs) {
        if (!firstReg) os << ",\n";
        firstReg=false;
        os << "    {\\"id\\":\"\" << json_escape(to_base_n(reg, cfg.base,
cfg.widthReg)) << "\",\\"addresses\\":[\n";
        bool firstAddr=true;
        for (auto& [addr, val] : addrs) {
            if (!firstAddr) os << ",\n";
            firstAddr=false;
            os << "      {\\"id\\":\"\" << json_escape(to_base_n(addr, cfg.base,
cfg.widthAddr))
                << "\",\\"value\\":\"\" << json_escape(val) << "\"}";
        }
        os << "\n      ]}\n";
    }

    os << "\n    ]\n";
    os << "}\n";
    return os.str();
}

// -----
// Offline plugin host (manifest: plugins/<name>/plugin.json)
// Input: input.json + code.txt; Output: output.json
// -----
struct Plugin {
    std::string name;
    std::string entry_win;
    std::string entry_lin;
    fs::path dir;
};

static std::string json_get_str(const std::string& j, const std::string& key){
    auto p = j.find("\" + key + "\"");
    if (p==std::string::npos) return {};
    p = j.find(':', p); if (p==std::string::npos) return {};
    p = j.find('"', p); if (p==std::string::npos) return {};
    auto q = j.find('"', p+1); if (q==std::string::npos) return {};
    return j.substr(p+1, q-(p+1));
}

static std::vector<Plugin> discover_plugins(const fs::path& pluginsRoot){
    std::vector<Plugin> out;
    fs::path root = pluginsRoot;
    if (root.empty() || !fs::exists(root)) return out;
    for (auto& e : fs::directory_iterator(root)) {

```

```
    if (!e.is_directory()) continue;
    auto dir = e.path();
    auto mf = dir / "plugin.json";
    if (!fs::exists(mf)) continue;
    std::string j; if (!read_all(mf, j)) continue;
    Plugin p;
    p.dir = dir;
    p.name = json_get_str(j, "name");
    p.entry_win = json_get_str(j, "entry_win");
    p.entry_lin = json_get_str(j, "entry_lin");
    if (!p.name.empty()) out.push_back(std::move(p));
}
return out;
}

static const Plugin* find_plugin(const std::vector<Plugin>& ps, const std::string& name){
    for (auto& p : ps) if (p.name==name) return &p;
    return nullptr;
}

static bool plugin_run(const Config& cfg, Workspace& ws, const Plugin& p,
                      long long bank, long long reg, long long addr,
                      const std::string& stdin_json_or_path,
                      std::string& report){

    Resolver R{cfg, ws};
    std::string raw;
    if (!R.get_value(bank, reg, addr, raw)) { report = "No value at that cell.";
return false; }
    std::string code = R.resolve_string(raw, bank, {});

    std::string bankStr = std::string(1, cfg.prefix) + to_base_n(bank, cfg.base,
cfg.widthBank);
    std::string regStr = to_base_n(reg, cfg.base, cfg.widthReg);
    std::string addrStr = to_base_n(addr, cfg.base, cfg.widthAddr);

    fs::path outdir = cfg.paths.out / "plugins" / bankStr /
("r"+regStr+"a"+addrStr) / p.name;
    fs::create_directories(outdir);

#ifndef _WIN32
    std::string entry = p.entry_win;
#else
    std::string entry = p.entry_lin;
#endif
    if (entry.empty()) { report = "Manifest missing entry for this OS."; return
false; }

    fs::path entryPath = fs::absolute(p.dir / entry);
    if (!fs::exists(entryPath)) { report = "Entry not found: " +
entryPath.string(); return false; }

    fs::path codeFile = fs::absolute(outdir / "code.txt");
```

```

fs::path inputFile = fs::absolute(outdir / "input.json");
fs::path outputFile = fs::absolute(outdir / "output.json");
fs::path logFile = fs::absolute(outdir / "run.log");
fs::path errFile = fs::absolute(outdir / "run.err");

if (!write_all(codeFile, code)) { report = "Cannot write code.txt"; return false; }

std::string stdin_json = "{}";
if (!stdin_json_or_path.empty()) {
    if (fs::exists(stdin_json_or_path)) { std::string tmp; if
(read_all(stdin_json_or_path, tmp)) stdin_json = tmp; }
    else stdin_json = stdin_json_or_path;
}

std::ostringstream is;
is << "{\n";
is << "  \"bank\": \"\" << json_escape(bankStr) << "\",\n";
is << "  \"reg\": \"\" << json_escape(regStr) << "\",\n";
is << "  \"addr\": \"\" << json_escape(addrStr) << "\",\n";
is << "  \"title\": \"\" << json_escape(ws.banks.at(bank).title) << "\",\n";
// Engine config for plugins (offline, no AI)
is << "  \"cfg\": {\n";
is << "    \"prefix\": \"\" << json_escape(std::string(1, cfg.prefix)) <<
"\",\n";
is << "    \"base\": " << cfg.base << ",\n";
is << "    \"widthBank\": " << cfg.widthBank << ",\n";
is << "    \"widthReg\": " << cfg.widthReg << ",\n";
is << "    \"widthAddr\": " << cfg.widthAddr << ",\n";
is << "    \"defaultReg\": " << cfg.defaultReg << "\n";
is << "  },\n";
// Absolute path to the context file for this bank
is << "  \"context_file\": \"\" << json_escape(fs::absolute(ctx_path(cfg,
bank)).string()) << "\",\n";
is << "  \"code_file\": \"\" << json_escape(codeFile.string()) << "\",\n";
is << "  \"stdin\": " << (stdin_json.empty()?"{}":stdin_json) << "\n";
is << "}\n";
if (!write_all(inputFile, is.str())) { report = "Cannot write input.json";
return false; }

int ec = 0;

#ifndef _WIN32
auto dq = [] (const std::string& s){ return "\"" + s + "\""; };
const std::string inner =
    dq(entryPath.string()) + " " + dq(inputFile.string()) + " " +
dq(fs::absolute(outdir).string()) +
    " > " + dq(logFile.string()) + " 2> " + dq(errFile.string());
const std::string cmd = std::string("cmd.exe /S /C ") + "\"" + inner + "\"";
ec = std::system(cmd.c_str());
#else
auto sq = [] (const std::string& s){ return "'" + s + "'"; };
const std::string inner =
    "\'" + entryPath.string() + "\' \'\' + inputFile.string() + "\' \'\' +

```

```

fs::absolute(outdir).string() +
    "\" > \" + logFile.string() + "\" 2> \" + errFile.string() +\"";
const std::string cmd = std::string("/bin/sh -c ") + sq(inner);
ec = std::system(cmd.c_str());
#endif

std::string outJson;
if (!read_all(outputFile, outJson)) {
    std::string errtxt; (void)read_all(errFile, errtxt);
    report = "Plugin did not produce output.json (exit=" + std::to_string(ec)
+ ")" + (errtxt.empty()?"":("\nerr:\n"+errtxt));
    return false;
}

std::string logtxt, errtxt;
(void)read_all(logFile, logtxt);
(void)read_all(errFile, errtxt);

std::ostringstream rep;
rep << "exit=" << ec << "\n";
if (!logtxt.empty()) rep << "log:\n" << logtxt << "\n";
if (!errtxt.empty()) rep << "stderr:\n" << errtxt << "\n";

report = rep.str();
return true;
}

// -----
// CLI
// -----
static void print_help(){
    std::cout <<
R"(
Linecraft CLI – Help (non-AI baseline)

```

---

Quick start

```

:open x00001
:ins 0001 hello
:insr 02 0003 world
:show
:w
:resolve
:export
:plugins
:plugin_run echo 01 0001 {"note":"demo"}
:q

```

## Commands

```

:help
:open <ctx>
:switch <ctx>
:preload
:ls
:show

```

```
:ins <addr> <value...>           (default reg = 01)
:insr <reg> <addr> <value...>
:del <addr>
:delr <reg> <addr>
:w
:resolve
:export
:plugins
:plugin_run <name> <reg> <addr> [stdin.json|inlineJSON]
:q

)" << "\n";
}

int main(int argc, char** argv){
    init_console_utf8();
    Config cfg;
    cfg.paths = make_runtime_paths(argc > 0 ? argv[0] : nullptr);
    fs::create_directories(cfg.paths.files);
    fs::create_directories(cfg.paths.out);
    Workspace ws;
    std::vector<Plugin> plugins = discover_plugins(cfg.paths.plugins);

    std::optional<long long> current;
    bool dirty=false;

    std::cout << "Linecraft CLI – shared core (non-AI)\n";
    std::cout << "Root: " << cfg.paths.root.string() << "\n";
    std::cout << "Type :help for commands.\n\n";

    auto ensure_current = [&](){
        if (!current) { std::cout << "No current context. Use :open <ctx>\n";
return false; }
        return true;
    };

    auto open_ctx = [&](const std::string& token){
        std::string t = token;
        if (!t.empty() && t[0]==cfg.prefix) t = t.substr(1);
        long long id=0;
        if (!parse_int_base(t, cfg.base, id)) { std::cout << "Bad ctx id\n";
return false; }

        auto p = ctx_path(cfg, id);
        if (fs::exists(p)) {
            std::string text, err;
            if (!read_all(p, text)) { std::cout << "Cannot read\n"; return false;
}
            Bank b;
            if (!parse_bank_text(text, cfg, b, err)) { std::cout << "Parse error:
" << err << "\n"; return false; }
            ws.banks[id] = std::move(b);
            ws.files[id] = p;
            current = id;
        }
    };
}
```

```
    dirty=false;
    std::cout << "Loaded " << p.string() << "\n";
    return true;
}

// create
Bank b;
b.id = id;
b.title = std::string(1,cfg.prefix) + to_base_n(id, cfg.base,
cfg.widthBank);
b.regs[cfg.defaultReg] = {};
auto text = write_bank_text(b, cfg);
if (!write_all(p, text)) { std::cout << "Cannot create file\n"; return
false; }
ws.banks[id] = std::move(b);
ws.files[id] = p;
current = id;
dirty=false;
std::cout << "Created " << p.string() << "\n";
return true;
};

auto preload = [&](){
fs::path dir = cfg.paths.files;
if (!fs::exists(dir)) return;
for (auto& e : fs::directory_iterator(dir)) {
    if (!e.is_regular_file()) continue;
    auto p = e.path();
    if (p.extension() != ".txt") continue;
    std::string text; if (!read_all(p, text)) continue;
    Bank b; std::string err;
    if (!parse_bank_text(text, cfg, b, err)) continue;
    ws.banks[b.id] = std::move(b);
    ws.files[b.id] = p;
}
std::cout << "Preloaded " << ws.banks.size() << " contexts.\n";
};

auto show = [&](){
if (!ensure_current()) return;
std::cout << write_bank_text(ws.banks[*current], cfg);
};

auto save = [&](){
if (!ensure_current()) return;
auto p = ctx_path(cfg, *current);
auto text = write_bank_text(ws.banks[*current], cfg);
if (!write_all(p, text)) std::cout << "Save failed\n";
else { dirty=false; std::cout << "Saved " << p.string() << "\n"; }
};

auto list = [&](){
if (ws.banks.empty()) { std::cout << "(no contexts)\n"; return; }
for (auto& [id,b] : ws.banks) {
```

```
        std::cout << cfg.prefix << to_base_n(id, cfg.base, cfg.widthBank) << "
(" << b.title << ")";
    if (current && *current==id) std::cout << " [current]";
    std::cout << "\n";
}
};

std::string line;
while (true) {
    std::cout << ">> ";
    if (!std::getline(std::cin, line)) break;
    auto s = trim(line);
    if (s.empty()) continue;

    if (s==":help") { print_help(); continue; }
    if (s==":preload") { preload(); continue; }
    if (s==":ls") { list(); continue; }
    if (s==":show") { show(); continue; }
    if (s==":w") { save(); continue; }

    if (s==":resolve") {
        if (!ensure_current()) continue;
        auto txt = resolve_bank_to_text(cfg, ws, *current);
        auto outp = out_resolved_path(cfg, *current);
        if (!write_all(outp, txt)) std::cout << "Write failed\n";
        else std::cout << "Wrote " << outp.string() << "\n";
        continue;
    }

    if (s==":export") {
        if (!ensure_current()) continue;
        auto js = export_bank_to_json(cfg, ws, *current);
        auto outp = out_json_path(cfg, *current);
        if (!write_all(outp, js)) std::cout << "Write failed\n";
        else std::cout << "Wrote " << outp.string() << "\n";
        continue;
    }

    if (s==":plugins") {
        plugins = discover_plugins(cfg.paths.plugins);
        if (plugins.empty()) std::cout << "(no plugins)\n";
        else for (auto& p : plugins) std::cout << " - " << p.name << " @ " <<
p.dir.string() << "\n";
        continue;
    }

    if (s==":q") {
        if (dirty) {
            std::cout << "Unsaved changes. Type :w to save or :q again to
quit.\n";
            std::string l2;
            std::cout << ">> ";
            if (!std::getline(std::cin, l2)) break;
            if (trim(l2)==":q") break;
        }
    }
}
```

```
        s = trim(l2);
    } else break;
}

// Tokenize
std::istringstream is(s);
std::vector<std::string> tok;
for (std::string t; is>>t;) tok.push_back(t);
if (tok.empty()) continue;

if (tok[0]==":open" && tok.size()>=2) { open_ctx(tok[1]); continue; }
if (tok[0]==":switch" && tok.size()>=2) { open_ctx(tok[1]); continue; }

if (tok[0]==":ins" && tok.size()>=3) {
    if (!ensure_current()) continue;
    long long addr=0; if (!parse_int_base(tok[1], cfg.base, addr)) {
std::cout << "Bad address\n"; continue; }
    auto pos = s.find(tok[2]);
    std::string value = (pos==std::string::npos)?"":s.substr(pos);
    ws.banks[*current].regs[cfg.defaultReg][addr] = value;
    dirty=true;
    continue;
}

if (tok[0]==":insr" && tok.size()>=4) {
    if (!ensure_current()) continue;
    long long reg=0, addr=0;
    if (!parse_int_base(tok[1], cfg.base, reg)) { std::cout << "Bad
reg\n"; continue; }
    if (!parse_int_base(tok[2], cfg.base, addr)) { std::cout << "Bad
addr\n"; continue; }
    auto pos = s.find(tok[3]);
    std::string value = (pos==std::string::npos)?"":s.substr(pos);
    ws.banks[*current].regs[reg][addr] = value;
    dirty=true;
    continue;
}

if (tok[0]==":del" && tok.size()>=2) {
    if (!ensure_current()) continue;
    long long addr=0; if (!parse_int_base(tok[1], cfg.base, addr)) {
std::cout << "Bad addr\n"; continue; }
    auto& m = ws.banks[*current].regs[cfg.defaultReg];
    auto n = m.erase(addr);
    std::cout << (n?"Deleted.\n":"No such addr.\n");
    if (n) dirty=true;
    continue;
}

if (tok[0]==":delr" && tok.size()>=3) {
    if (!ensure_current()) continue;
    long long reg=0, addr=0;
    if (!parse_int_base(tok[1], cfg.base, reg)) { std::cout << "Bad
reg\n"; continue; }
```

```
        if (!parse_int_base(tok[2], cfg.base, addr)) { std::cout << "Bad
addr\n"; continue; }
        auto itR = ws.banks[*current].regs.find(reg);
        if (itR==ws.banks[*current].regs.end()) { std::cout << "No such
reg\n"; continue; }
        auto n = itR->second.erase(addr);
        std::cout << (n?"Deleted.\n":"No such addr.\n");
        if (n) dirty=true;
        if (itR->second.empty()) ws.banks[*current].regs.erase(itR);
        continue;
    }

    if (tok[0]==":plugin_run" && tok.size()>=4) {
        if (!ensure_current()) continue;
        auto* pl = find_plugin(plugins, tok[1]);
        if (!pl) { std::cout << "No such plugin. Use :plugins\n"; continue; }
        long long reg=0, addr=0;
        if (!parse_int_base(tok[2], cfg.base, reg)) { std::cout << "Bad
reg\n"; continue; }
        if (!parse_int_base(tok[3], cfg.base, addr)) { std::cout << "Bad
addr\n"; continue; }
        std::string stdin_json = "{}";
        if (tok.size()>=5) {
            auto pos = s.find(tok[4]);
            stdin_json = (pos==std::string::npos)?"{}":s.substr(pos);
        }
        std::string rep;
        bool ok = plugin_run(cfg, ws, *pl, *current, reg, addr, stdin_json,
rep);
        std::cout << (ok?"Plugin ok\n":"Plugin failed\n") << rep << "\n";
        continue;
    }

    std::cout << "Unknown / malformed command. Type :help\n";
}

return 0;
}
```