OSAWARE BASIC

Version 7 — Architecture Reference

Build 1775412127 · April 2026

1. Executive Summary

OSAWARE V7 is a fully browser-based operating system that gives anyone with a web browser access to a **threaded, memory-safe, multi-process computing environment** — no installation, no configuration, no prior programming experience required. Write a program, press RUN, and your application executes in its own isolated process with access to 3D graphics, audio synthesis, real-time networking, and the world's leading AI — all from a language as simple and readable as plain English.

> **The vision:** *A complete operating system that runs everywhere, programmes like 1980s BASIC, thinks like 2026 AI, and scales from a single line of code to a full multi-window application suite.*

What OSAWARE V7 Delivers

A Real Operating System in Your Browser

V7 is not a toy interpreter or a scripting sandbox. It is a layered operating system with a genuine kernel, a process table, per-process isolated memory, a typed syscall bus connecting six hardware drivers, and a virtual file system with 53 built-in programs. When you run a program, it gets its own PID, its own memory heap, and its own execution context — entirely separate from the shell that launched it. When the program ends, the OS reclaims its resources and returns cleanly to the terminal. This is how professional operating systems work, and it now runs in a browser tab.

True Multi-Window Multi-Tasking

V7 introduces a full **Window IPC system** — the ability to launch any BASIC program as a first-class OS process in its own browser window, tracked by the kernel, and reachable from the parent terminal at any time. Launch MAZE3DV2 in a new window while your terminal session continues uninterrupted. Send messages to it. Receive replies. Check its status. Close it. All with three lines of BASIC:

```
pid = LAUNCH("MAZE3DV2")        '  open in new window, get PID
WINDOW.SEND pid, "LEVEL:5"       '  send a message to it
ON WINDOW GOSUB HandleReply      '  fire when it replies
```

The terminal always knows what is running. Type MEM at any time and see every active process — in-terminal programs and windowed apps together — with their PIDs, names, and status. Close a window and it disappears from the table automatically. This is the OSAWARE process model: simple enough to learn in five minutes, powerful enough to build real applications.

World-Class Graphics and Audio

OSAWARE V7 includes a full Three.js WebGL 3D engine with physically-based rendering (PBR), normal maps, roughness maps, environment maps, fog, point lights, and chrome reflections — accessible through simple single-line BASIC commands. The 2D pixel engine renders directly to GPU-accelerated canvas with a sprite and collision system for game development. The audio system supports multi-voice synthesis with custom waveforms, harmonic layering, and synchronised playback queuing. All of this from a language that looks like this:

```
GL.INIT : GL.SPHERE 1
GL.NORMALMAP meshId, "stone"
GL.ROUGHNESS meshId, 0.3
GL.DRAW meshId
```

AI as a First-Class Language Feature

V7 is the first BASIC dialect to treat **AI as a native data type**. The AI command sends a prompt to Claude and streams the response directly into the terminal — or captures it silently into a string variable. The AINUM command returns a numeric AI response. Both are usable anywhere in a BASIC program, making Claude a **programmable variable** inside any application a user writes:

```
10 AI "Describe this planet in one sentence", D$
20 PRINT "The AI says: "; D$
```

```
10 AINUM "Rate the user's last move, 1 to 10", Score
20 IF Score > 7 THEN PRINT "Excellent!"
```

> **The roadmap:** V7 establishes the foundation. Future builds will deepen the AI integration — making it possible to bind AI responses to game state, use Claude as a non-player character, generate level data on the fly, describe an image and receive BASIC code that draws it, or build conversational applications where the AI and the user co-author the program in real time. The combination of **simple, readable BASIC syntax** with **access to the world's most capable AI** creates something

> genuinely new: a programming environment where the barrier to creation is not technical skill — it is imagination.

V7 at a Glance

Capability	Detail
Operating system	Layered kernel, 6 hardware drivers, typed syscall bus, process table, per-process isolated memory
Multi-tasking	Per-program PIDs, ProcessMemory isolation, Window IPC with BroadcastChannel messaging
BASIC language	187 commands across 17 categories — flow, data, I/O, graphics, audio, net, AI, IPC, 3D
3D graphics	Three.js WebGL — PBR materials, point lights, fog, chrome, mesh builder, 40+ commands
2D graphics	GPU-accelerated pixel engine, image store, sprite/collision system
Audio	Multi-voice synthesis, custom waveforms, harmonic layering, synchronised queuing
Networking	WebSocket client — real-time data streams, GOSUB event handlers
AI integration	Claude AI as native BASIC commands — stream to terminal or capture to variable
File system	53 built-in programs, user program persistence, VFS asset folders, VFSPUT/VFSGET
Windowed apps	LAUNCH, WINDOW.SEND/REPLY, ON WINDOW GOSUB — full IPC between OS windows
Browser native	No install, no server required — runs from a local file or any static web host
Open & extensible	Clean driver architecture — new runtimes (Lua, WASM) can be added without touching the kernel

> **Build 1775412127 · April 2026** · 16,494 lines across 18 JS files · 99/99 regression tests passing · 53 built-in programs

2. File Structure & Line Counts

All JavaScript source files, ordered by layer. Build stamp is embedded in every file via the cache-busting version parameter in index.html.

File	Lines	Role	Layer

core/constants.js	36	Global constants (MAX_LINES, CMD_OK, ASS_* types)	Foundation
core/libraries.js	77	LinePrinter, History classes	Foundation
core/kernel/bus.js	110	KernelBus — typed syscall router (post/call/on/emit)	Kernel
core/kernel/process_memory.js	124	ProcessMemory — 56-property isolated process heap	Kernel
core/kernel/kernel.js	132	Kernel — process table, PID registry, scheduler	Kernel
core/shell.js	1,286	ShellRuntime — REPL, MEM, HELP, windowed launch	Runtime
core/vfs.js	4,962	Virtual File System — 53 programs + user assets	Services
core/drivers/terminal.js	1,163	TerminalDriver — DOM, keyboard, cursor, colour, I/O	Drivers
core/drivers/gl3d.js	1,214	GL3DDriver — Three.js WebGL 3D (62 methods)	Drivers
core/drivers/gfx.js	996	GfxDriver — 2D pixel engine, image store, sprites	Drivers
core/drivers/audio.js	214	AudioDriver — SOUND/WAVE/BEEP synthesis	Drivers
core/drivers/net.js	135	NetDriver — WebSocket (WS.OPEN/SEND/RECV)	Drivers
core/drivers/window.js	320	WindowDriver — cross-window IPC, PID tracking	Drivers
core/kernel.js	3,908	Interpreter — BASIC runtime, 187 commands, tick loop	Runtime
core/compiler.js	1,443	Expression compiler — eval, lookup_, assign_	Runtime
core/boot.js	374	Boot — DOM setup, URL params, resize, oEmulator init	Bootstrap
core/ui.js	~60	UI — fullscreen/overscan click handlers	Bootstrap

Script Loading Order

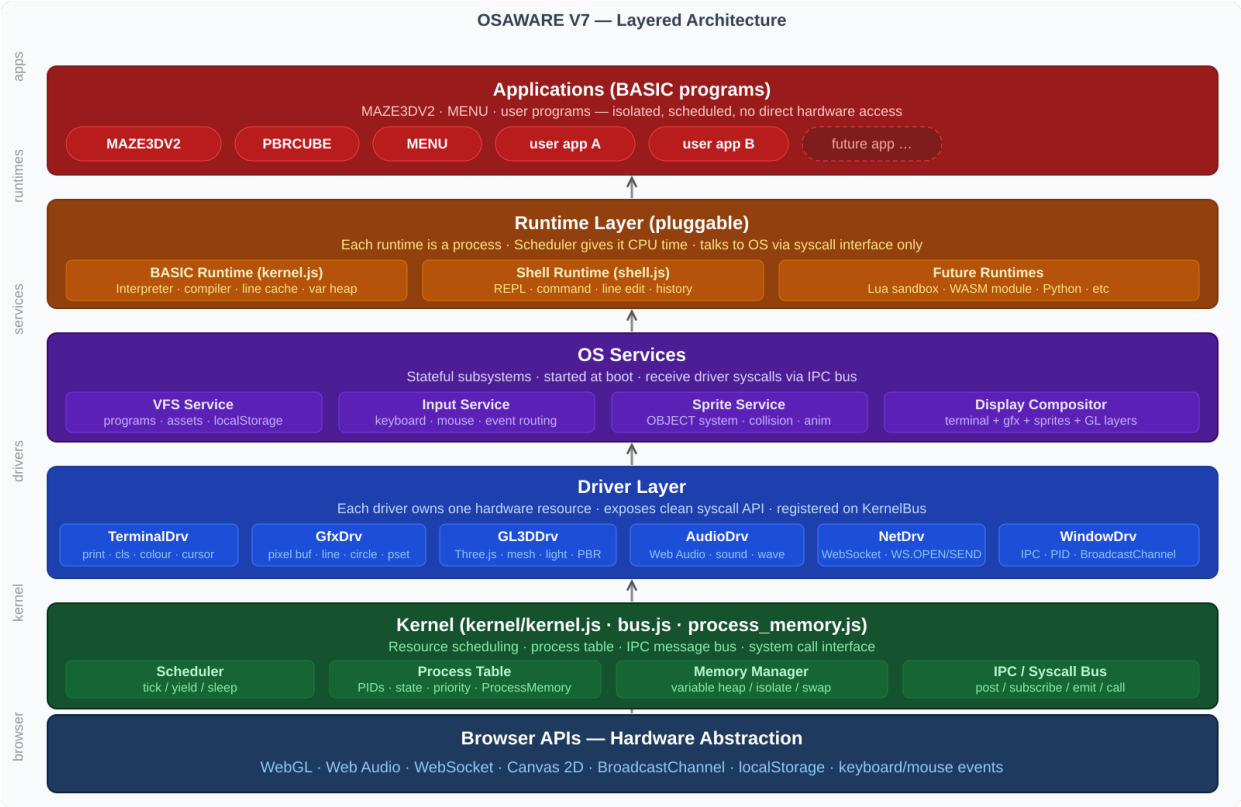
The order in index.html is critical — each layer depends on the previous:

```
1.  three.min.js          — Three.js (vendor, no version stamp)
2.  constants.js          — global constants must load first
3.  kernel/bus.js         — KernelBus class
4.  kernel/process_memory.js — ProcessMemory class
5.  kernel/kernel.js      — Kernel class (depends on ProcessMemory)
6.  shell.js              — ShellRuntime (depends on Kernel)
7.  libraries.js          — LinePrinter, History
```

```
8.  vfs.js                — VirtualFS
9.  drivers/terminal.js   — TerminalDriver
10. drivers/gl3d.js       — GL3DDriver
11. drivers/gfx.js        — GfxDriver
12. drivers/audio.js      — AudioDriver
13. drivers/net.js        — NetDriver
14. drivers/window.js     — WindowDriver
15. kernel.js             — Interpreter (depends on all drivers)
16. compiler.js           — mixed into Interpreter prototype
17. boot.js               — creates oEmulator, starts tick loop
18. ui.js                 — attaches DOM button handlers
```

3. Layered Architecture

OSAWARE V7 implements a strict six-layer OS model. Each layer communicates only with adjacent layers — no layer reaches past its boundary.



Layer	Name	Components	Communication
L0	Hardware (Browser APIs)	DOM, WebGL context, AudioContext, WebSocket, BroadcastChannel, window.open	Native browser APIs

L1	Kernel	kernel/kernel.js — process table kernel/bus.js — syscall bus kernel/process_memory.js — memory	bus.on() / bus.post() / bus.call()
L2	Drivers	terminal.js, gl3d.js, gfx.js audio.js, net.js, window.js	Registered on KernelBus Host getters/setters for cross-driver state
L3	OS Services	vfs.js — Virtual File System libraries.js — LinePrinter, History	Called directly by runtime No bus dependency
L4	Runtimes	kernel.js — BASICRuntime (Interpreter) shell.js — ShellRuntime (REPL)	BASIC commands → bus.post() Shell → Interpreter host ref
L5	Applications	53 BASIC programs in VFS User-saved programs in localStorage	BASIC language surface only No direct JS access

Dependency Rules

- L1 (Kernel) has zero dependencies — it is constructed first
- L2 (Drivers) depend on L0 (browser) and post to L1 bus — never call each other directly
- L3 (Services) are standalone — they do not know about the bus or drivers
- L4 (Runtime) depends on all lower layers — it wires everything together
- L5 (Applications) can only use the BASIC language surface — zero direct JS

4. Kernel & KernelBus

4.1 Kernel Class (core/kernel/kernel.js — 132 lines)

The Kernel maintains the process table and allocates ProcessMemory for each registered process. It does not implement scheduling — the Interpreter's tick loop drives execution.

Method / Property	Description
`registerProcess(proc, priority)`	Allocates a fresh ProcessMemory, registers process, assigns PID. Calls proc.attachMemory(mem). First registered process becomes PID 1 (shell).
`spawnProcess(proc, priority)`	Alias for registerProcess — future hook for true fork()
`getMemory(pid?)`	Returns the ProcessMemory for the given PID (or active PID)
`listProcesses()`	Returns array of {pid, name, state, priority, lines} — used by MEM command
`activeProcess()`	Returns the process record for _activePid
`kill(pid?) / pause(pid?) / resume(pid?)`	Update process state in the table

`_nextPid`	Auto-incrementing PID counter. Shell = PID 1, in-terminal programs start at PID 2, window processes start at PID 100
`_processes`	Map<pid, {pid, process, memory, state, priority, name}>

4.2 KernelBus (core/kernel/bus.js — 110 lines)

The KernelBus is the message-passing backbone of the OS. Every BASIC command that uses a driver goes through the bus. This creates a hard interface boundary — drivers can be swapped, mocked, or upgraded without changing the runtime.

API: `bus.on(syscall, handler)` · `bus.post({syscall, param})` · `bus.call({syscall, ...})` · `bus.emit(event)` · `bus.listen(event, fn)`

Syscall group	Examples	Driver
`gl.*`	gl.init, gl.cls, gl.sphere, gl.texture, gl.draw, gl.emissive, gl.normalmap, gl.envmap, gl.pointlight … (40+ syscalls)	GL3DDriver
`gfx.*`	gfx.pset, gfx.line, gfx.circle, gfx.fillrect, gfx.paint, gfx.loadimg, gfx.display, gfx.imglist, gfx.imgfree	GfxDriver
`audio.*`	audio.sound, audio.wave, audio.beep, audio.soundwait, audio.soundresume	AudioDriver
`net.*`	net.open, net.send, net.close, net.onmsg, net.status, net.clear	NetDriver
`window.*`	window.launch, window.send, window.close, window.reply, window.onmsg, window.status	WindowDriver
`program.*`	program.stop (lifecycle event emitted by _onProgramStop)	All drivers listen

Data flow: BASIC command → Driver

```
BASIC program: "SOUND 440, 60"
  → Interpreter.interpret("SOUND 440, 60")
  → cmdSOUND("440, 60")
  → this.kernel.post({syscall:"audio.sound", param:"440, 60"})
  → KernelBus routes to AudioDriver.cmdSOUND("440, 60")
  → AudioDriver creates OscillatorNode, schedules playback
```

5. Process Memory & Isolation

5.1 ProcessMemory (core/kernel/process_memory.js — 124 lines)

Each process gets an isolated ProcessMemory instance. The Interpreter accesses its state entirely through proxy getters/setters that delegate to this._mem, making memory context switching a single swapMemory() call.

Category	Properties	Contents
Program store	5 properties	lines[] (10,000 slots), lines_assigned (Set), _lineCache (compiled), _labels (Map), _subs (Map)
Variable heap	5 properties	variables_numbers (Map), variables_strings (Map), variables_arr_numbers (Map), variables_arr_strings (Map), variables_func []
Flow control	13 properties	gosubs[], gosub_level, fors[], for_level, for_var, _if_stack, _select_stack, _while_stack, _sub_stack, _in_sub, _shared_vars, _static_vars, _on_goto_table
Execution state	12 properties	run_line, running, if_line, line_remaining, just_stopped, data[], data_count, data_position, _error_trap_line, _error_resume_line, _last_err, _in_error
BASIC heap	5 properties	_memory (Uint8Array 64KB), _optionBase, _dimInfo, _arrMax, _func_result
Runtime flags	4 properties	_trace, _rng_seed, want_input, want_ai

Total: 56 properties, all accessed through 88 proxy getter/setter pairs on the Interpreter. Zero direct field access from outside the memory context.

5.2 Per-Program PID Allocation

Every RUN command allocates a fresh ProcessMemory and registers a new kernel PID:

```
Shell starts            → PID 1  (ProcessMemory: shell)
RUN MAZE3DV2            → PID 2  (ProcessMemory: MAZE3DV2 program)
MAZE3DV2 ends           → PID 2 unregistered, swapMemory back to PID 1
RUN PBRCUBE             → PID 3  (ProcessMemory: PBRCUBE program)
RUN TESTS -w            → PID 100 (WindowDriver registry, separate OS instance)
```

5.3 Memory Context Switch

swapMemory() on the Interpreter is the context switch primitive:

```
// run() — before starting a program:
const progMem = new ProcessMemory();
progMem.lines = this._mem.lines;  // transfer loaded program
this._programPid = os._nextPid++;
os._processes.set(pid, { pid, memory: progMem, name: progName, ... });
this.swapMemory(progMem);  // all property access now hits progMem
```

```
// _onProgramStop() — after program ends:
os._processes.delete(this._programPid);
this.swapMemory(this._shellMem);  // restore shell context
```

6. Driver Reference

All drivers follow the same pattern: a class with a host reference (the Interpreter), host getter/setter forwarders for cross-cutting state, and methods registered on the KernelBus.

6.1 TerminalDriver (1,163 lines, 39 methods, 14 accessors)

Owns all terminal I/O. The Interpreter has zero DOM knowledge — everything routes through the terminal driver.

Owned state: output div, canvas, colour palette (17 entries), cursor, cols/rows, keyboard input state, history, line printer, mouse state, screen buffer (LOCATE mode), 67 total properties

Host forwards: running, run_line, interpret(), tick(), assign_(), _fireEventGosub(), _gfxFlush() — 13 forwarders

Graphics references: _gfx → GfxDriver._gfx (live getter), _glCanvas → GL3DDriver._glCanvas (live getter), _spr → Interpreter.__spr

6.2 GL3DDriver (1,214 lines, 62 methods, 9 accessors)

Full Three.js WebGL wrapper. 40+ BASIC commands expose scene setup, mesh building, PBR materials, and per-frame rendering.

Command group	BASIC commands
Scene setup	GL.INIT, GL.CLS, GL.PERSPECTIVE, GL.CAMERA, GL.LOOKAT
Render modes	GL.WIRE, GL.SOLID, GL.SOLIDWIRE
Lighting	GL.LIGHT, GL.AMBIENT, GL.POINTLIGHT, GL.LIGHTSOFF
Material (Phong)	GL.COLOUR, GL.SHINE, GL.ALPHA, GL.EMISSIVE, GL.WIRECOLOR
PBR material maps	GL.NORMALMAP, GL.ROUGHMAP, GL.AOMAP, GL.HEIGHTMAP, GL.METALMAP, GL.EMISSIVEMAP, GL.ROUGHNESS, GL.METALNESS, GL.EMISSIVEINTENSITY, GL.ENVMAP
Mesh building	GL.BEGIN, GL.VERTEX, GL.FACE, GL.END, GL.SPHERE, GL.BOX, GL.MESHID
Textures	GL.TEXTURE, GL.CHROME
Transforms	GL.TRANSLATE, GL.ROTATE, GL.SCALE
Visibility	GL.HIDE, GL.SHOW, GL.CLOSE

Atmosphere	GL.FOG, GL.FOGOFF
Rendering	GL.DRAW, GL.DRAWALL

6.3 GfxDriver (996 lines, 51 methods, 55 accessors)

2D pixel engine using a CPU-side Uint8Array buffer uploaded to a WebGL DataTexture each frame. Includes an image store (load/display/free) and the sprite/object animation system.

Commands: PSET, PRESET, LINE, CIRCLE, FILLCIRCLE, RECT, FILLRECT, PAINT, POINT, LOADIMG, DISPLAY, IMGLIST, IMGFREE, OBJECT.* (10 commands), COLLISION, ON COLLISION

6.4 AudioDriver (214 lines, 11 methods)

Web Audio API wrapper. Supports multi-voice synthesis with custom waveforms, synchronised playback queuing, and real-time waveform upload.

Commands: SOUND freq,dur[,vol[,voice]], SOUND WAIT, SOUND RESUME, WAVE voice,SIN|array, BEEP

6.5 NetDriver (135 lines, 17 methods, 11 accessors)

WebSocket client. Supports message queuing, GOSUB event handlers, and connection state polling.

Commands: WS.OPEN url$, WS.SEND msg$, WS.CLOSE, WS.CLEAR, WS.ONMSG lbl · Functions: WS.RECV$, WS.STATUS

6.6 WindowDriver (320 lines, 21 methods, 7 accessors)

Cross-window IPC system. Opens child OSAWARE windows as managed kernel processes with bidirectional messaging.

Component	Detail
Transport	postMessage (parent → child, via stored window reference) + BroadcastChannel "osaware-ipc" (child → parent). Both channels used; _seen deduplication set prevents double-handling.
PID allocation	Window PIDs start at 100, incrementing. In-terminal program PIDs start at 2. Shell is always PID 1.
Child detection	_initAsChild() runs if ?ppid= URL param is present. Registers message listener, stores opener reference, defers "ready" broadcast by 300ms (DOM timing).
Close detection	500ms closePoller checks win.closed. beforeunload also fires "closed" message. Dedup _seen set prevents double notification.
MEM integration	cmdMEM() reads both os.listProcesses() and winDrv._windows, pruning closed entries live.
Pre-validation	RUN -w and LAUNCH check VFS _files, _userFiles, and XHR HEAD ./files/PROG.bas before calling window.open() — blank windows never open for missing programs.

7. Window IPC System

The Window IPC system allows BASIC programs running in separate browser windows to communicate as first-class OS processes. Each child window is a fully independent OSAWARE instance — its own Interpreter, Kernel, drivers, and ProcessMemory.

7.1 BASIC Surface

Command / Variable	Side	Description
`pid = LAUNCH("PROGNAME")`	Parent	Open child window, validate program exists first, return PID
`RUN PROGNAME -w`	Parent	Shell shorthand — same as LAUNCH from the prompt
`WINDOW.SEND pid, msg$`	Parent	Send string message to child window via postMessage
`WINDOW.CLOSE pid`	Parent	Send kill signal and close child window
`WINDOW.STATUS(pid)`	Parent	0 = closed, 1 = running, 2 = starting
`ON WINDOW GOSUB lbl`	Both	Fire GOSUB handler when a message arrives
`WINDOW.MSG$`	Both	String variable: contents of last received message
`WINDOW.PID`	Both	Numeric: PID of the window that sent the last message
`WINDOW.REPLY msg$`	Child	Send string back to parent via BroadcastChannel
`WINDOW.ISCHILD$`	Child	"1" if this window was opened by LAUNCH, "0" otherwise

7.2 Message Flow

```
Parent: pid = LAUNCH("MYGAME")
  → WindowDriver validates program in VFS/_files/_userFiles/XHR
  → window.open("index.html?run=MYGAME&ppid=1&cpid=100")
  → WindowDriver registers {pid:100, win, status:0, name:"MYGAME"}
  → closePoller starts (500ms interval, checks win.closed)

Child boot:
  → boot.js reads ?run=MYGAME → _initCmd = "RUN MYGAME"
  → WindowDriver._initAsChild() reads ?ppid=1, ?cpid=100
  → 300ms later: _sendToParent({type:"ready", pid:100, ppid:1})
  → Parent terminal prints: [OSAWARE] MYGAME (PID 100) running.

Parent sends: WINDOW.SEND 100, "LEVEL:5"
  → postMessage to child window ref
  → Child _handleIncoming({type:"send", msg:"LEVEL:5"})
  → Child fires ON WINDOW GOSUB handler
```

```
  → WINDOW.MSG$ = "LEVEL:5"

Child replies: WINDOW.REPLY "ACK:LEVEL:5"
  → postMessage to opener + BroadcastChannel.postMessage
  → _seen dedup set prevents double-handling
  → Parent fires ON WINDOW GOSUB handler
  → WINDOW.MSG$ = "ACK:LEVEL:5", WINDOW.PID = 100

Child window closed:
  → closePoller detects win.closed → status=0
  → _handleIncoming({type:"closed", pid:100})
  → Parent terminal prints: [OSAWARE] MYGAME (PID 100) closed.
```

7.3 URL Parameter Protocol

Child windows are opened with a structured URL that encodes the IPC relationship:

```
index.html?run=PROGNAME&ppid=PARENT_PID&cpid=CHILD_PID
```

run=Program to auto-run on startup (passed as initCmd to Interpreter)
ppid=Parent PID — stored in child WindowDriver as _parentPid
cpid=Child PID — assigned by parent WindowDriver before opening window

8. BASIC Language Surface

187 registered commands across all categories. The Interpreter builds two lookup structures: _commandTable (Map<string, fn>) for O(1) lookup, and _commandMap (array) for ordered matching of multi-word commands.

Category	Commands
Program flow	GOTO, GOSUB, RETURN, IF/THEN/ELSE/ELSEIF/END IF, FOR/NEXT, WHILE/WEND, BREAK, CONT, END, STOP, SWITCH/CASE/DEFAULT/END SWITCH
Data	DIM, ERASE, CLEAR, DATA, READ, RESTORE, LET, SWAP, OPTION BASE, POKE, POKEW, POKEL
I/O	PRINT, PRINT USING, INPUT, LINE INPUT, WRITE, LPRINT, LLIST, LOCATE, COLOUR/COLOR, CLS, RESET
File system	LOAD, SAVE, MERGE, NEW, LIST, DIR, FILES, VFSPUT, VFSGET$, VFSIMG, VFSDEL, DELUSER
Subroutines	SUB/END SUB, FUNCTION/END FUNCTION, CALL, SHARED, STATIC, EXIT SUB, EXIT FUNCTION, DECLARE, RETURN
Error handling	ON ERROR GOTO, RESUME, RESUME NEXT, ERR, ERL

2D Graphics	PSET, PRESET, LINE, CIRCLE, FILLCIRCLE, RECT, FILLRECT, PAINT, POINT, WIDTH, RESIZE
Images	LOADIMG, DISPLAY, IMGLIST, IMGFREE
Sprites	OBJECT.SHAPE, OBJECT.ON/OFF, OBJECT.X/Y, OBJECT.VX/VY, OBJECT.AX/AY, OBJECT.START/STOP, OBJECT.CLOSE, OBJECT.PRIORITY, OBJECT.HIT, OBJECT.CLIP, OBJECT.PLANES, COLLISION ON/OFF/STOP, ON COLLISION GOSUB
3D / WebGL	GL.INIT through GL.DRAWALL (40+ commands — see Section 6.2)
Audio	SOUND, SOUND WAIT, SOUND RESUME, WAVE, BEEP
Networking	WS.OPEN, WS.SEND, WS.CLOSE, WS.CLEAR, WS.ONMSG
Window IPC	LAUNCH, WINDOW.SEND, WINDOW.CLOSE, WINDOW.REPLY, ON WINDOW GOSUB
Mouse	MOUSE ON/OFF/STOP, ON MOUSE GOSUB
AI	AI, AINUM, AIKEY, AICLEAR
View	FULLSCREEN ON/OFF, OVERSCAN ON/OFF
Debug/Shell	TRON, TROFF, MEM, INFO, HISTORY, LABELS, GLDEBUG, EDIT, DELETE
System	SLEEP, DELAY, BEEP, RANDOMIZE, REM, DECLARE

Numeric Functions

ABS, SGN, INT, FIX, SQR, SIN, COS, TAN, ATN, EXP, LOG, RND, VAL, LEN, CLNG, CSNG, LBOUND, UBOUND, INSTR, PEEK, CSRLIN, ERL, ERR, UPTIME, SECONDS, TIMER, COLS, ROWS, WIDTH, HEIGHT, KEYDOWN(n), MOUSE(n), POINT(x,y), COLLISION(id), WS.STATUS, WINDOW.STATUS(pid), WINDOW.PID, LAUNCH("prog"), GL.MESHID

String Functions

MID$, LEFT$, RIGHT$, UPPER$, LOWER$, UCASE$, STR$, CHR$, ASC, INSTR, SPACE$, STRING$, HEX$, OCT$, CENTER$, TAB$, LINES$, DATE$, TIME$, WS.RECV$, WINDOW.MSG$, WINDOW.ISCHILD$, VFSGET$(path)

9. Virtual File System

The VFS (core/vfs.js — 4,962 lines) is the OS's storage layer. It provides a unified namespace for programs, assets, and user data with three backing stores and transparent URL loading.

Store	Backing	Contents
System store	Embedded JS array (vfs.js)	53 BASIC programs compiled inline. Always wins on name collision. Zero network requests.
User store	localStorage	User-saved programs and assets. Persist across

		sessions. Keyed as osaware_user_files.
Disk store	./files/ via XHR	Fallback for programs in the files/ directory. Async load triggers _runAfterLoad flag.
Assets	Embedded + user assets	Folders: MAZE3D/ (STONE.PNG, FLOOR.PNG, CEIL.PNG), DEMO/. User assets via VFSPUT/VFSIMG.

Installed Programs (53 total)

AASNOOPY, ANIMAL, AUDIOTEST, BALLS, BENCH, BLEH, BLOB, CASETEST, CIRCLES, COL, COUNT, DATATEST, FRAC, GFXTEST, GFXTESTS, GL3D, GLDEMO, GLDEMOMAX, GLSHADE, HUGO, IMG, IMG2, IMGDEMO, LABELDEMO, LINES, LINES2, LOGO, LOGODEV, LOOPTEST, MANDEL, MAZE3D, MAZE3DV2, MENU, MMIND, MOUSE, MOUSEDEMO, PACMAN, PBRCUBE, PROFILER, PSETTEST, QUINE, SGIDEMO, SNAKE, SSH, STARS1, STARS2, TESTS, TODO, VALUHACK, VFSTEST, VFSTEST2, WUMPUS

10. Key Design Patterns & Lessons

10.1 Host Getter/Setter Forwarding

Every driver stores a reference to the Interpreter as this._host. Cross-cutting state (colours, running flag, terminal output) is accessed via getters/setters that proxy to the owning object. This avoids circular references while giving drivers transparent access.

```
// In GfxDriver constructor:
get running() { return this._host.running; }
set running(v) { this._host.running = v; }
appendLine(t,n) { return this._host.appendLine(t,n); }
```

Critical rule: Every property forwarded must have BOTH get and set, even if the driver never writes it. A getter-only property throws a silent runtime error when the driver writes to it.

10.2 ProcessMemory Proxy Pattern

The Interpreter exposes its 56 memory properties as getter/setter pairs that delegate to this._mem. swapMemory() replaces _mem in a single assignment — all subsequent property access transparently uses the new context.

```
// In Interpreter (kernel.js):
get variables_numbers()  { return this._mem.variables_numbers; }
set variables_numbers(v) { this._mem.variables_numbers = v; }
// ... 88 such pairs total

swapMemory(mem) { const old = this._mem; this._mem = mem; return old; }
```

10.3 initCmd Timing

The Interpreter constructor receives an initCmd string (e.g., "RUN MAZE3DV2" from ?run= URL param). It must NOT execute in the constructor — the DOM is not ready. It is stored as _initCmd and executed inside execute() after setup() completes, with a 50ms setTimeout to ensure DOM layout has settled.

10.4 Deduplication for Dual-Channel IPC

BroadcastChannel and postMessage can both deliver the same message, causing double handling. The WindowDriver maintains a _seen Set keyed on type:pid:msg:timestamp. Any message matching a key already in _seen is discarded. Entries expire after 2 seconds to prevent memory growth.

10.5 double-underscore Escape Hatch

Some proxy getters and driver-owned fields share names. The _spr (sprite scene) and _gfx (GFX scene) properties are stored directly on the Interpreter as this.__spr and this.__gfx (double underscore) to avoid colliding with the terminal driver's forwarding chain.

10.6 Program Stop Event Chain

When a program ends, three things must happen in order: (1) TerminalDriver._onProgramStop() resets colours and hides GFX/sprite canvases, (2) kernel.emit("program.stop") fires so GL3DDriver hides its canvas via the bus listener, (3) the program PID is unregistered and shell memory is restored. The _onProgramStop delegate in kernel.js handles all three.

11. Build & Deployment

11.1 Build System

OSAWARE V7 has no build step — all JavaScript is served as-is. Cache busting is handled by a Unix timestamp query parameter appended to every script tag in index.html. The build number is updated by the development toolchain when any source file changes.

```
<!-- index.html — all scripts load with version stamp -->
<script src="core/kernel.js?v=1775412127"></script>
```

Current build: `1775412127` (Unix timestamp — April 2026)

11.2 Deployment

The distribution is a single ZIP archive containing:

- core/ — all JavaScript source files
- files/ — 52 BASIC programs as plain .bas text files (one per program)
- docs/ — documentation assets
- index.html — single-page application entry point
- htaccess — Apache configuration for serving locally

No server-side processing required. Can be served from any static host or opened directly as a local file (file:// protocol). WebSocket features require a reachable WebSocket server.

11.3 Content Security Policy

ui.js is a separate file specifically to satisfy strict CSP. Inline event handlers (onclick=) are not permitted. All DOM event listeners are attached via addEventListener in ui.js after DOMContentLoaded.

11.4 Browser Requirements

- WebGL 1.0 (for 3D GL commands)
- Web Audio API (for SOUND/WAVE)
- WebSocket (for WS.* commands)
- BroadcastChannel (for Window IPC — all modern browsers)
- localStorage (for user program persistence)
- ES2020 or later (classes, async/await, optional chaining)

12. V6R2 → V7 Migration Summary

Metric	V6R2	V7
kernel.js lines	7,508	3,908 (−48%)
Total JS files	5	18
Total JS lines	~12,000	16,494
Driver files	0	6 (terminal, gl3d, gfx, audio, net, window)
Kernel layer files	0	3 (bus, process_memory, kernel)
BASIC commands	~170	187
Process isolation	None — single shared state	ProcessMemory per program run
PIDs	None	PID 1 (shell) + PID per run + PID 100+ (windows)

Cross-window IPC	None	BroadcastChannel + postMessage + WindowDriver
VFS programs	47	53
Regression tests	99/99	99/99
Duplicate methods	Several (silent JS override)	0 (code review pass)
Constructor timing bugs	initCmd fired before DOM	Fixed: deferred to execute()