Activerse Developers Insider Release: 2

Engine: Activerse v1.4.2
Window title: `Activerse Instance v1.4.2`
Date of this note: August 2026 (engine tagged 30 July 2026)
Companion: ActiWiki · JavaDoc · Version logs

This is the successor to *Activerse Developers Insider Release: 1*, which described the v1.0 / v1.1.2-era engine (World, Actor, input, sound, debug, End button). Release 2 is the same kind of document for the current tree: two packages, a monotonic game clock, interpolated rendering, classpath assets, camera, config, ACEHS, and the utils layer that accumulated through the v1.4 series.

The wiki is the tutorial. This file is the architecture brief: how the pieces connect, why the clock is written the way it is, and what changed between Insider 1 and v1.4.2.

Table of contents

1. Overview
2. How a session starts
3. Package map
4. The monotonic clock
5. World
6. Actor
7. Camera
8. Input
9. Images, sound, and ResourcePaths
10. Collision
11. Config
12. Debug, logging, and ACEHS
13. ActiverseUtils
14. How it all connects
15. Lineage from the versions page
16. Practical notes
17. Conclusion

Overview

Activerse is a Java 2D game / simulation / physics engine. You subclass `Actor` and `World`, then call `Activerse.start(world)` from `Main.java`. The engine owns the window, the dual-thread game loop, input, images, sound, optional lighting, and a debug overlay. `ActiverseUtils` is the toolbox that grew around that core: paths, physics, particles, save/load, UI drawing, ACEHS formatting.

v1.4.2 is backwards compatible with the v1.4.* series. Simulation code should keep using `getX()` / `getY()`. Drawing that wants smoothness should use the render helpers (`getRenderX()`, `getRenderY()`, and friends).

You can check shipped defaults in the codebase.

How a session starts

1. Run `Main.java`. The template ships with `Activerse.start(...)` commented out; uncomment it with your world.
2. `Activerse.start(World)` runs on the EDT: builds a non-resizable, centered `JFrame`, packs the world, then constructs `GameLoop` and starts it.
3. `ConfigPuller` loads `Activerse.properties` from the classpath (never from `settings.ini` for engine keys).
4. `World` construction sizes the panel as `width * cellSize` by `height * cellSize` pixels, optionally adds the Debug button (`show_debug`), and always adds End (`Activerse.shutdownApplication()`).
5. Each tick, `World.update()` snapshots every actor, then calls `act()` on anyone who is not tick-inert. The render thread samples the clock, writes render alpha, and `repaint()`s.

World switching (`Activerse.setWorld`) stops the previous loop, saves if the world implements halt hooks, swaps the panel on the EDT, and starts a new `GameLoop`. That path was hardened in v1.4.1 / v1.4.2 so teardown cannot deadlock the update thread.

Package map

ActiverseEngine

Class	Role
`Activerse`	Lifecycle: start, stop, setWorld, shutdown
`World`	JPanel world: actors, paint, debug, lighting, pause
`Actor`	Entities: act, move, paint, inventory, interpolation snapshots
`GameLoop`	Dual-thread update/render on a monotonic nanosecond schedule
`Camera`	Viewport follow, zoom, interpolated world↔screen
`ConfigPuller`	Engine properties + optional `settings.ini` game keys
`KeyboardInfo`	Key state, edges, alphanumeric text helpers
`ActiverseMouseInfo`	Mouse buttons and position (volatile component ref)
`ActiverseImage` / `ActiverseSound`	Assets via ResourcePaths
`CollisionManager`	Pixel-perfect overlap with a soft image cache
`Item`	Inventory payload
`MemoryTracker`	MPS, heap, optional `logs.log`

ActiverseUtils

`ResourcePaths`, `ErrorLogger`, `Physics`, `ActorVector`, `MathUtils`, `PerlinNoise`, `WorldGeneration`, `WorldManager`, `WorldKeys`, `ParticleSystem`, `Shaper`, `TextLabel`, `UIBar`, `PanelPainter`, `TextRenderUtils`, `ImageUtils`, `ColorUtils`, `FileUtils`, `JsonUtils`, `IniUtils`, `Timer`, `DelayScheduler`, `DaemonExecutors`.

Import the engine with `import ActiverseEngine.*;`. Pull utils by name.

The monotonic clock

This is the v1.4.2 headline. Insider 1 described a Swing `Timer` driving the world. v1.0.9 introduced `GameLoop` (“Hypersmooth”). v1.4.2 rewrote the schedule so **update and render share one monotonic clock**.

Why `System.nanoTime()`, not wall time (major shifts)

`System.currentTimeMillis()` is wall-clock time. It jumps when the OS steps the clock. `System.nanoTime()` is a **monotonic** elapsed-time source: it only moves forward, which is what you want for frame pacing. We had synchronization issues with complex games and wanted to fix that.

`GameLoop` stores:

- `FRAME_TIME_NANOS = 1_000_000_000L / TARGET_FPS` (default 60 → ~16.67 ms)
- `epoch = System.nanoTime()` captured when the update loop starts
- `lastUpdateTimeNanos` — the **scheduled** time of the last completed tick, not “when `update()` returned”

Using the scheduled tick time (not end-of-update) keeps interpolation from sitting one tick behind when the two threads are phase-locked.

Update loop (epoch ticks)

tickNumber = (now - epoch) / FRAME_TIME_NANOS

- If `tickNumber` has not advanced, the thread sleeps until the next boundary (`Thread.sleep` for >2 ms, `LockSupport.parkNanos` for shorter waits).
- If the process stalls and several tick numbers are skipped, **those ticks are dropped**. The loop does not catch up with a burst of `update()` calls. That is the spiral-of-death guard: under load you lose simulation steps instead of compounding hitch + catch-up hitch.
- After a successful tick: `lastUpdateTimeNanos = epoch + currentTickNumber * FRAME_TIME_NANOS`. Drift cannot accumulate because every deadline is `epoch + n * frame`.

`World.update()` is skipped when a subclass returns true from `isSimulationDrivenBySwingTimer()` (legacy / EDT-owned simulation). Default worlds are driven by this loop.

Render loop (time sampler)

Render does not own a second clock. Each pass:

1. `alpha = clamp((now - lastUpdateTimeNanos) / FRAME_TIME_NANOS, 0, 1)`
2. `world.setRenderAlpha(alpha)`
3. `world.repaint()`
4. Sleep the remainder of the frame budget, same sleep strategy as update

`alpha` is the fraction of the way from the previous simulation snapshot toward the current one. Actor paint blends with it. Logic still uses integer (or precise) simulation coordinates.

What you should call

Context	**API**
Movement, collision, AI	`getX()`, `getY()`, precise setters if you use them
Custom `paint`	`getRenderX()`, `getRenderY()`, `getRenderPreciseX/Y()`, `getRenderDirection()`, `getRenderBoundingBox()`
Camera-aware draw	`Camera.worldToScreenInterpolated(x, y, world.getRenderAlpha())`
Target FPS	`fps` in `Activerse.properties`

Do not mix wall-clock (`currentTimeMillis`) into the game loop. `ActiverseUtils.Timer` and particle lifetimes still use millis for gameplay delays; that is separate from frame pacing.

World

`World` extends `JPanel` and still implements `ActionListener` / `KeyListener`. Constructor: `super(cellsWide, cellsHigh, cellSize)` → pixel size `cells * cellSize`.

Important methods (Insider 1 plus what landed later):

- `setBackgroundImage`, `addObject`, `removeObject`, `addSound`
- `start` / `stop`, `pause` / `resume` (v1.3.2)
- `showText`
- `getRenderAlpha` / `setRenderAlpha` (loop-owned)
- `paintComponent`: background → optional lighting → actors (interpolated paint) → text → debug overlay

Dynamic lighting (v1.2 experimental, overhauled v1.4.0, buffer reuse v1.4.2) composites through a `BufferedImage` matched to world size so the buffer is not allocated every frame.

Static vs tick-inert: `Actor.setStatic(true)` hides the actor from the debug list (v1.4.0). Override `isTickInert()` to skip `act()` entirely — useful for thousands of tiles.

Focus: the world disables tab traversal so Swing buttons (Debug, End) do not steal letter keys. Those buttons are `setFocusable(false)`.

Actor

Still the entity base: location, image, `act()`, `paint()`, `move`, `turn`, `keyIsDown(char)`.

v1.4.2 paint uses interpolated position/direction. Each tick begins with `snapshotState()` so previous and current pose exist for blending.

Other surfaces that did not exist in Insider 1:

- Velocity (`get/setVelocityX/Y`): v1.1.1, for DIY gravity
- `getWidth` / height: v1.3.0
- `useAStar(Actor other, int iterations:` v1.3.0
- Inventory: `enableInventory`, `addItem`, `removeItem`, `setInventorySize` — v1.4.0
- Precise (sub-pixel) position flags
- `delayNext` via `DelayScheduler`

`keyIsDown(char)` is case-sensitive. Prefer `KeyboardInfo.isLetterDown('w')` for WASD.

Camera

Added in v1.4.1 for large / infinite maps. `World` does **not** construct a camera; you own one.

- `new Camera(viewW, viewH, smoothness)`
- `setTarget(actor)`, `setWorldBounds`, `setClampToWorldBounds(false)` for open maps
- `update()` each tick
- `zoomIn` / `zoomOut` / `setZoom` (clamped ~0.6–1.4)
- `isVisible(...)` including an interpolated overload
- `worldToScreen` / `screenToWorld` / `worldToScreenInterpolated`

Pair camera offsets with `World.getRenderAlpha()` so the viewport and actors interpolate on the same clock. Made pretty easy for beginners but allows advanced features too.

Input

KeyboardInfo

Still a 256-slot down array plus KeyListener. v1.4.x additions:

- `isKeyDown(int | char | "shift")`
- `isLetterDown`, `isAnyKeyDown`
- `justPressed` / `justReleased` (you pass current and previous samples)
- `appendAlphaNumeric` + `syncAlphaNumericState` for name fields (sync when entering a menu so held keys do not dump characters)

World keeps keyboard focus; Debug/End do not take it.

ActiverseMouseInfo

Replaced the fragile v1 mouse path. v1.4.1 added a component reference so coordinates match the world panel. v1.4.2 made that reference `volatile` for cross-thread reads.

Images, sound, and ResourcePaths

`ActiverseImage` and `ActiverseSound` resolve through `ResourcePaths`:

1. If the path is an existing filesystem file, use that URL.
2. Else treat it as a classpath key (leading slashes stripped) so `java -jar` works.

MediaTracker uses a shared `Canvas` peer. Sound streams close on stop; `finalize()` was removed (JDK cleanup in v1.4.2). Volume is `setVolume(float 0..1)` / `getVolume()` from v1.3.2. Pause/resume exist on the sound object.

Prefer keys like `assets/images/player.png` and keep the same layout on disk and in the jar.

Collision

`CollisionManager.intersects` is pixel-perfect when both actors have loaded images: bounding-box intersection, then non-transparent pixel overlap. Missing images fall back to boxes.

v1.4.2 caches `Image → BufferedImage` in a `WeakHashMap` of `SoftReference`s so conversion is not repeated every collision test, without pinning images forever.

Config

`ConfigPuller` (v1.4.1, synchronized load in v1.4.2):

- Engine keys **only** from classpath `Activerse.properties`: `fps`, `show_debug`, `logging`, `dynamicLighting` / `dynamic_lighting`
- Other keys may come from `settings.ini` via `IniUtils` (`game.*` style)
- `saveEngineSettings(...)` writes the engine toggles to `src/Activerse.properties` if `src` exists, else the working-directory file. It does not rewrite `settings.ini`.

Missing properties file: ACEHS `PROP.IN.OUT.IO` and built-in defaults (`show_debug` defaults **true** in code as well as in the shipped file).

Debug, logging, and ACEHS

Overlay

`show_debug=true` (the v1.4.2 default) adds a Debug button. Overlay shows FPS, ticks, MPS, non-static actors + collision flag, loaded images (duplicate counts, MIA if missing on classpath), playing sounds, active keys. Syntax errors are still compile-time. Runtime issues often also print ACEHS lines.

End is always present.

logs.log

`logging=true` → `MemoryTracker` appends one line per second (heap, non-heap, GC time, FPS vs target, system millis, tick interval). Session header is a timestamp. Logs do not include actor/image/sound lists.

ACEHS

Introduced v1.3.1, centralized in `ErrorLogger` by v1.4.2:

FILE.TYPE:(LN: methodSignature - ACEHS Error thrown; message)

FILE.TYPE-CONNTO-OTHER.TYPE:(LN: methodSignature - ACEHS Error thrown; message)

`LN` is a method name, not a source line. Engine codes are `1A`–`13A` plus `PROP`. Utils are `1B`… plus live `WM` from `WorldManager`.

ActiverseUtils

Short map of why these exist (most arrived in v1.4.0 thru 1.4.2):

Cluster	**Classes**	**Origin**
Motion / math	`Physics`, `ActorVector`, `MathUtils`	1.4.0 & 1.4.1
Terrain	`PerlinNoise`, `WorldGeneration`	1.4.0

Cluster	Classes	Origin
Persistence	`WorldManager`, `WorldKeys`, `FileUtils`, `JsonUtils`, `IniUtils`	1.4.x
FX / HUD	`ParticleSystem`, `Shaper`, `TextLabel`, `UIBar`, `PanelPainter`, `TextRenderUtils`	1.4.1&1.4.2
Assets / color	`ResourcePaths`, `ImageUtils`, `ColorUtils`	1.4.1&1.4.2
Timing / threads	`Timer`, `DelayScheduler`, `DaemonExecutors`	1.4.0 / 1.4.2
Errors	`ErrorLogger`	1.3.1 idea, 1.4.2 home

`WorldManager` stores worlds under a `Worlds/` directory resolved from cwd or its parent (project root vs `src` as working directory). Treat it as advanced.

Lineage from the versions page

This is what sits between Insider 1 (v1.0 engine) and this release.

Version	Date	What it added that still matters
1.1.2	Jul 2024	Log sessions, independent FPS calc, RAM pass; *Insider 1 research PDF*
1.2.0	Jul 2024	Dynamic lighting (early), logging system time
1.2.1	Sep 2024	Heap monitors, mouse detection experiments

Version	Date	What it added that still matters
1.3.0	Sep 2024	JavaDoc site, mouse rewrite, `getWidth`, A*, constructor leak fix
1.3.1	Nov 2024	ACEHS, null safety, ticks as long, JDK20+ lambdas
1.3.2	Jan 2025	Sound volume, pause/resume, ticks in debug, 60 FPS default, Breakout demo
1.4.0	Jun 2025	Perlin worlds, physics helpers, lighting overhaul, collision revamp, multithreading loop, items/inventory, static actors, Timer/FileUtils/ActorVector
1.4.1	Mar 2026	Camera, particles, Shaper, ConfigPuller, Math/Color utils, UIBar/text bars, world-switch fixes, unified error handling
1.4.2	Jul 2026	**This release:** unified monotonic clock, skipped missed ticks, actor/camera interpolation, ResourcePaths, collision image cache, reused lighting buffer, KeyboardInfo helpers, EDT-safe loop attach, JDK cleanup

v1.4.2 states compatibility with v1.4.* as a “no port” solution (that is, update game engine and leave it be). Older notes claimed ports back toward 1.0.7 / 1.0.3-theta; **do not expect a 1.0 project to drop in without package and API work.**

Practical notes (AI-Gen)

First world

```
public class Main {

    public static void main(String[] args) {

        Activerse.start(new MyWorld());

    }

}
```

`MyWorld` → `super(400, 400, 1)`, `addObject(new Player(), 100, 100)`. Player `setImage(new ActiverseImage("player.png"))`, move from `act()` with `KeyboardInfo.isLetterDown`.

Do

- Run `Main.java` only.
- Draw with render helpers; simulate with `getX`/`getY`.
- Keep images/sounds on a path ResourcePaths can see (file or classpath).
- Override `isTickInert()` on scenery.

Do not

- Catch up missed ticks yourself in a way that fights GameLoop's skip policy.
- Assume debug is off — it is on in the shipped properties.
- Copy Insider 1's "add a second KeyListener to World" pattern unless you have a reason; sample `KeyboardInfo`.
- Share `logs.log` casually (timing and heap figures).

Conclusion

Insider 1 was World + Actor + input + sound + a debug button. v1.4.2 is still that engine, with a clock that does not drift, a render thread that only samples, assets that survive jar packaging, and a utils package so you are not glued to third-party math/UI/save code.

The contract for game code is unchanged: subclass, `act()`, `Activerse.start`. The contract for *smooth* games is new: interpolate on the monotonic alpha, and leave simulation on the tick.