LCM Nav3D

True Volumetric 3D Navigation for Unreal Engine

User Manual

Version 2.2.0 • Unreal Engine 5.8.0

Generated from the documentation shipped inside the plugin.
Every chapter is also available as markdown in `Plugins/LCMNav3D/Documentation/`.

Contents

Part I

Getting Started

CHAPTER 1

Tutorial 01: Installation

TL;DR: Put the plugin in `YourProject/Plugins/LCMNav3D/`, open the project and let it compile, enable **Show Plugin Content** in the Content Browser, then open a demo map and press Play. If the demo map works, your install is good.

Time: ~10 minutes (plus one compile).

1.1 Before you start

Requirement	Detail
Unreal Engine	5.8
Platform	Windows 64-bit or Linux
Project type	**C++ project** for a manual install (see below)
Compiler	Visual Studio 2022 with the *Game development with C++* workload (Windows)

The plugin ships as source, so something has to compile it once. Which "something" depends on how you install it.

1.2 Method A: Installed from Fab / Epic Games Launcher

This is the simplest path and works with **Blueprint-only projects**.

1. Install the plugin from the Launcher's **Library** → **Fab Library** section. It lands in `<Engine>/Plugins/Marketplace/LCMNav3D/` and is already compiled for 5.8.
2. Open your project.
3. **Edit** → **Plugins**, search "LCM Nav3D", tick **Enabled** if it isn't already.
4. Restart the editor when prompted.

Skip to Verify the install.

1.3 Method B: Manual install into your project

Use this when you have the plugin as a folder (for example from a direct purchase or a version you re-targeted yourself).

1. Close the editor.
2. Copy the `LCMNav3D` folder into your project so the layout is:

```
YourProject/
  YourProject.uproject
  Plugins/
    LCMNav3D/
      LCMNav3D.uplugin
```

```
Source/
Content/
Config/
Shaders/
```

> Create the `Plugins` folder if your project doesn't have one. The folder must be named exactly `LCMNav3D`, because the content mount `/LCMNav3D/` depends on it.

3. **If your project is Blueprint-only, convert it to C++ first.** A Blueprint-only project has no build pipeline and cannot compile a source plugin. The easiest conversion: **Tools → New C++ Class → None → Create Class**. The editor generates a `Source/` folder and restarts. You never have to write any C++ afterwards.
4. Right-click `YourProject.uproject` → **Generate Visual Studio project files**.
5. Open the generated `.sln` and build **Development Editor | Win64**, or open the `.uproject` and click **Yes** when the editor offers to rebuild.
6. The plugin is `EnabledByDefault`, so it turns itself on. Confirm under **Edit → Plugins**.

1.4 Verify the install

Do all three of these. Each one catches a different kind of failure.

1.4.1 1. The plugin loaded

Edit → Plugins → AI Navigation category. You should see **LCM Nav3D**, enabled, version 2.2.0.

1.4.2 2. You can see the plugin's content

Plugin content is hidden by default in the Content Browser:

Content Browser → Settings (the gear icon) → tick "Show Plugin Content".

A **LCM Nav3D Content** folder appears in the tree. Without this you will not find any demo map, and it is by far the most common "the plugin didn't install" false alarm.

1.4.3 3. A demo map actually runs

Open **`LCM Nav3D Content → Demo → BehaviourTree → InfniteMaps → Vertical_Skyscraper_BT`** and press **Play**.

You should see agents flying through the building: over floors, through gaps, not along the ground. Give it a second or two on first play: the navigation volume builds at `BeginPlay`.

> **`Demo → MainMenu`** is a gallery of every demo in the plugin, if you would rather browse than open maps by name.

If that works, **your install is correct** and you can move on to Tutorial 02: Core Concepts.

1.5 The engine plugins it turns on

`LCMNav3D.uplugin` declares these as dependencies, so the engine enables them for you. You do not need to touch them, but it is worth knowing they are on:

Plugin	Why it's needed
`StateTree`, `GameplayStateTree`	The StateTree task/condition/evaluator set

Plugin	Why it's needed
`MassEntity`, `MassGameplay`, `MassAI`	The Mass crowd layer
`SmartObjects`	Smart Object behaviour definitions
`DataValidation`	In-editor validation of illegal Mass trait combinations

> **If the plugin fails to load with `GetLastError 126`**, one of the Mass plugins got disabled. `MassGameplay` in particular is required at runtime. Re-enable it and restart.

1.6 Common install problems

"The following modules are missing or built with a different engine version: LCMNav3D" The plugin hasn't been compiled for your engine build. Click **Yes** to rebuild. If the rebuild fails, you are on a Blueprint-only project, so go back to Method B step 3.

I can't find the demo maps. You haven't enabled *Show Plugin Content*. See verification step 2.

The plugin is missing from the Plugins window entirely. The folder is in the wrong place or misnamed. It must be `YourProject/Plugins/LCMNav3D/LCMNav3D.uplugin`. Check you didn't end up with a doubled folder like `Plugins/LCMNav3D/LCMNav3D/`.

Compile errors mentioning StateTree or Mass types. An engine plugin dependency was disabled. Re-enable the table above and regenerate project files.

More in Troubleshooting.

1.7 Other engine versions

This copy targets **Unreal Engine 5.8**. The plugin ships as a separate copy for each supported engine version, **5.2 through 5.8**, and each one is built and verified against that engine.

Install the copy that matches your project rather than re-targeting this one. Engine APIs move between versions, and the per-version copies already carry those differences.

Next: Tutorial 02: Core Concepts. What the navigation manager is actually doing, and the three settings that decide whether it works well in your level.

CHAPTER 2

Tutorial 02: Core Concepts

TL;DR: LCM Nav3D carves your level's **empty air** into a tree of boxes (a Sparse Voxel Octree) and paths through the empty ones. You place one **`LcmNavigationManagerSVO`** actor per level, tell it how big your world is and how small the tightest gap is, and it does the rest.

Time: ~10 minutes reading. No project changes. This is the chapter that makes every later setting make sense.

2.1 1. Why not just use Unreal's navmesh?

Unreal's Recast navmesh is a **surface**. It answers "where can I stand?" by flattening your level onto walkable polygons. That is exactly right for a soldier and completely wrong for a dragon, because the interesting space for a flyer is the part with nothing in it.

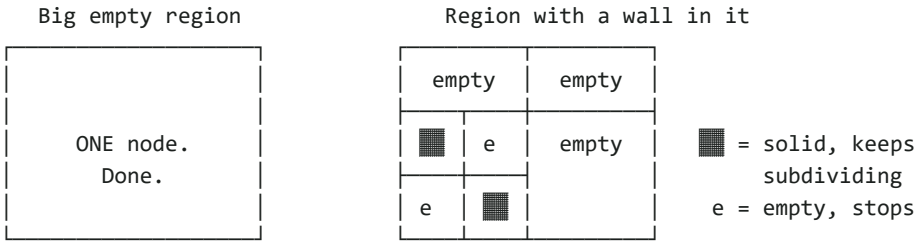
LCM Nav3D answers a different question: **"where is there empty air?"** It represents the volume, not the floor. That's the whole idea. Everything else is engineering around making that fast.

2.2 2. The Sparse Voxel Octree, in one picture

Imagine a giant cube around your level. Now:

1. Is the cube completely empty? **Yes** → done, one big box of free space.
2. **No**, something's in it? → cut it into 8 smaller cubes and ask each one the same question.
3. Repeat until the cubes reach your minimum size.

That's it. That's the octree.



"Sparse" is the important word. Open sky costs almost nothing: one node covers a huge volume. Detail only appears where geometry actually is. This is why a 2 km map doesn't cost 2 km worth of memory: most of it is air, and air is cheap.

What this means for you: the cost of your navigation data is driven by *how much clutter you have*, not by how big your level is.

2.3 3. The one actor you must place

`LcmNavigationManagerSVO`: drop one into your level. One per level, that's the rule.

It owns the octree, runs the pathfinder, and answers every query. Every agent talks to it. If it isn't in the level, nothing navigates.

Its Details panel is deliberately organised in the order you should think about it:

Category	What it's for
1. Setup	Finite or infinite world, the biggest decision
2. Voxels	Size and resolution
3. Collision	Which geometry counts as an obstacle
4. Performance	Frame-budget knobs
5. Debug	Visualisation
6. Runtime State	Read-only live info

Full field-by-field detail is in the Settings Reference. Here we only cover the concepts behind them.

2.4 4. Finite vs Infinite: the decision that shapes everything

This is **`1. Setup → Infinite World`**, and it changes which other settings even appear.

2.4.1 Finite (default, `Infinite World` unticked)

One fixed box of navigation, built once. You set **`WorldExtent`** (default **10000 cm** = a 200 m cube, since extent is measured from the centre outward).

- Built at `BeginPlay` if **`Auto Build On Play`** is on (it is by default).
- Simple, fast, completely predictable.
- **Use this for:** arenas, levels, dungeons, single buildings, anything with a boundary.

Start here. Most projects never need anything else.

2.4.2 Infinite (`Infinite World` ticked)

The world is divided into **chunks** of **`ChunkSize`** (default **8000 cm**). Chunks are generated around your agents as they move and thrown away behind them.

- **Use this for:** open worlds, streaming levels, procedural terrain, anything without a boundary.
- Costs more complexity: chunks must generate before an agent can route through them.

> **`ChunkSize` is not a free dial.** It decides both memory *and* whether distant routes can be found at all. **8000 is the tested default. Change it only with a reason and re-test.** Larger chunks mean fewer, heavier generations; smaller means more, lighter ones, and more seams to cross.

2.5 5. Resolution: the two settings people get wrong

2.5.1 `MinVoxelSize` (default 100 cm)

How small the smallest box is allowed to get. **This is the single biggest driver of both quality and cost.**

- **Smaller** → finds tighter gaps, uses more memory, takes longer to build.
- **Larger** → cheaper and faster, but narrow openings vanish and agents route around them.

> ⚠ **It's a floor, not an exact size.** The octree halves its way down, so the real leaf size is the world (or chunk) size repeatedly divided by two until it's about your `MinVoxelSize`. Two consequences that surprise people:
>
> - Changing `MinVoxelSize` from 100 to 90 may change **nothing** (same number of halvings).
> - Changing it from 100 to 60 may cross a threshold and **double your memory** in one step.
>
> Tune it by testing, not by assuming the number is literal.

Rule of thumb: set `MinVoxelSize` to roughly **half the narrowest gap** your agents must fly through. A 3 m window needs about 150 cm or finer.

2.5.2 `ClearancePadding` (default 100 cm)

Inflates obstacles by this much when voxelizing, so agents don't clip walls.

> ⚠ **The classic mistake: setting this too high seals your level.** If padding approaches the size of your voxels, doorways and corridors fill in completely and the pathfinder reports "no route" through an opening you can plainly see. **Keep `ClearancePadding` well below `MinVoxelSize`.** If you need more clearance than that, lower `MinVoxelSize` too.

If agents refuse to path through an opening, this setting is the first thing to check.

2.6 6. How a path is actually found

For a short hop, it's a straight search through the octree. For anything longer, there are two levels:

```
1. MACRO: "which chunks/regions do I cross?"     (coarse, cheap, long-range)
                  ↓
2. MICRO: "which voxels inside them?"            (fine, detailed, short-range)
                  ↓
3. SMOOTH: turn the blocky voxel path into a nice line
```

You don't call these yourself; it's automatic. The reason to know is that it explains a common symptom: if a long route fails but a short one works, the problem is usually **macro**. Either chunks are not generated yet, or `ChunkSize` is interacting badly with your layout.

2.6.1 Choosing a solver

Solver	Character	Use when
A* *(Fastest, Grid Locked)*	Cheapest; paths follow voxel edges, so turns look square	You'll smooth the result anyway, or you need max throughput
Theta* *(Accurate, Any Angle)*	Most direct; cuts corners properly	Quality matters more than cost
Lazy Theta* *(Balanced)*	Nearly Theta* quality, near A* cost	**The sensible default**

2.6.2 Smoothing

Mode	Result
Raw Path	Straight out of the solver, blocky
Linear Shortcut	Removes redundant waypoints; straight segments
Curved Spline	Smooth curves: **what you want for flying things**

> Smoothing runs *after* pathfinding and is validated against geometry, so a smoothed path won't cut through a wall.

2.6.3 Where the work runs

Path solving runs on the CPU, on a worker pool, off the game thread. It is deterministic: same input, same output, every time - which is what makes replays, lockstep multiplayer and automated tests reproducible.

The GPU is used for **voxelization**: turning your geometry into the octree, where it is substantially faster. If no suitable GPU is present it falls back to the CPU automatically, so nothing breaks.

2.7 7. Things that move

Static geometry is baked into the octree. For things that move, add a **`LcmDynamicObstacleComponent`** to the actor. It restamps the octree as the actor moves, and agents reroute around it. That's Tutorial 05.

2.8 8. Terms you'll meet in the rest of the docs

Term	Meaning
SVO	Sparse Voxel Octree, the navigation data
Voxel / node	One box in the octree; a **leaf** is a smallest one
Chunk	One tile of an infinite world
Portal	A connection between two chunks: how routes cross a seam
Macro / micro	Coarse long-range vs fine short-range pathfinding
Clearance	How much room an agent needs to fit
Flow field	One shared steering field many agents sample (for crowds)

2.9 Recap

- The octree stores **empty space**, and empty space is cheap.
- **One `LcmNavigationManagerSVO`** per level.
- **Finite** unless you genuinely have an open world.
- **`MinVoxelSize`** ≈ half your tightest gap; it snaps to powers of two.
- **`ClearancePadding`** below `MinVoxelSize`, or you'll seal your own level.
- **Lazy Theta*** + **Curved Spline** is a good default for a flyer.

Next: Tutorial 03: Your First Flying Agent. Build a working agent in your own level.

CHAPTER 3

Tutorial 03: Your First Flying Agent

TL;DR: Three ingredients: an **`LcmNavigationManagerSVO`** in the level, a **pawn with an AIController**, and a Behavior Tree whose only task is **`LCM Fly To (SVO)`** pointed at a blackboard key. No C++.

Time: ~20 minutes. **You'll end with:** a pawn that flies through 3D space to a goal in your own level.

3.1 What we're building

```
Level
  ├── LcmNavigationManagerSVO        ← the brain (one per level)
  ├── BP_MyFlyer  (Pawn)             ← the body
  │     └── AIController: BP_MyFlyerAI
  │             └── runs BT_MyFlyer
  │                     └── Blackboard: BB_MyFlyer  (Vector key "GoalLocation")
  └── TargetPoint                    ← where we want it to go
```

3.2 Step 1: Add the navigation manager

1. In the **Place Actors** panel, search **`LcmNavigationManagerSVO`**.
2. Drag one into your level. Position doesn't matter much. It uses its own location as the centre of the navigation volume, so put it roughly in the middle of your playable space.
3. Select it and set, in the Details panel:

Category	Field	Value
1. Setup	`Infinite World`	**unticked** (finite, see Tutorial 02)
1. Setup	`Auto Build On Play`	**ticked**
2. Voxels	`World Extent`	large enough to cover your play space (default **10000** = 200 m cube)
2. Voxels	`Min Voxel Size`	**100** to start
2. Voxels	`Clearance Padding`	**100** to start

> `World Extent` measures **outward from the actor**, so 10000 gives you a 200 m cube centred on the manager. If your agents will fly outside that box, increase it: outside the volume there is no navigation data and paths will fail.

3.2.1 See what it built

Tick **5. Debug** → **`Draw Debug Volumes`** and press Play. You'll see the octree drawn over your level: lots of big boxes in open air, small ones packed around geometry. **This is the single most useful thing you can do while tuning.** If a doorway you care about shows no small voxels, the pathfinder can't see it either.

Turn it back off when you're done; it's expensive to draw.

3.3 Step 2: Make the pawn

You need something that can move in 3D without gravity.

1. **Content Browser → Add → Blueprint Class → Pawn**. Name it **`BP_MyFlyer`**.
2. Open it and add a **Floating Pawn Movement** component. (Add Component → search "Floating Pawn Movement".) This is what makes it fly rather than fall.
3. Select `FloatingPawnMovement` and set **Max Speed** to **600**. Match this to the task's `Flight Speed` later.
4. Add a **Static Mesh** component so you can see it. A cone or sphere is fine. Scale it down to something sensible.
5. In **Class Defaults**, set **`Auto Possess AI`** to **`Placed in World or Spawned`**.

> **Collision tip.** For your first test, set the mesh's collision to **No Collision**. Physics fighting the navigation is a very common source of "my agent jitters / gets stuck", and you want a clean first result. Add collision back once flying works.

3.4 Step 3: Blackboard and Behavior Tree

3.4.1 The blackboard

1. **Add → Artificial Intelligence → Blackboard**. Name it **`BB_MyFlyer`**.
2. Add one key:
   - **Type: `Vector`**, **Name: `GoalLocation`**

> The Fly To task also accepts an **Object** key holding an Actor. Use a Vector for now; the Actor form is what you'd use for chasing, covered at the end.

3.4.2 The behavior tree

1. **Add → Artificial Intelligence → Behavior Tree**. Name it **`BT_MyFlyer`**.
2. Open it, and in the Details panel set **Blackboard Asset** to `BB_MyFlyer`.
3. Drag from **ROOT** and add a **Sequence**.
4. Drag from the Sequence and add the task named **`Fly To`**; it appears in the tree as **`LCM Fly To (SVO)`**.
5. Select that task and set:

Field	Value
Blackboard Key	`GoalLocation`
Acceptance Radius	`100`
Flight Speed	`600`

Leave everything else at defaults for now.

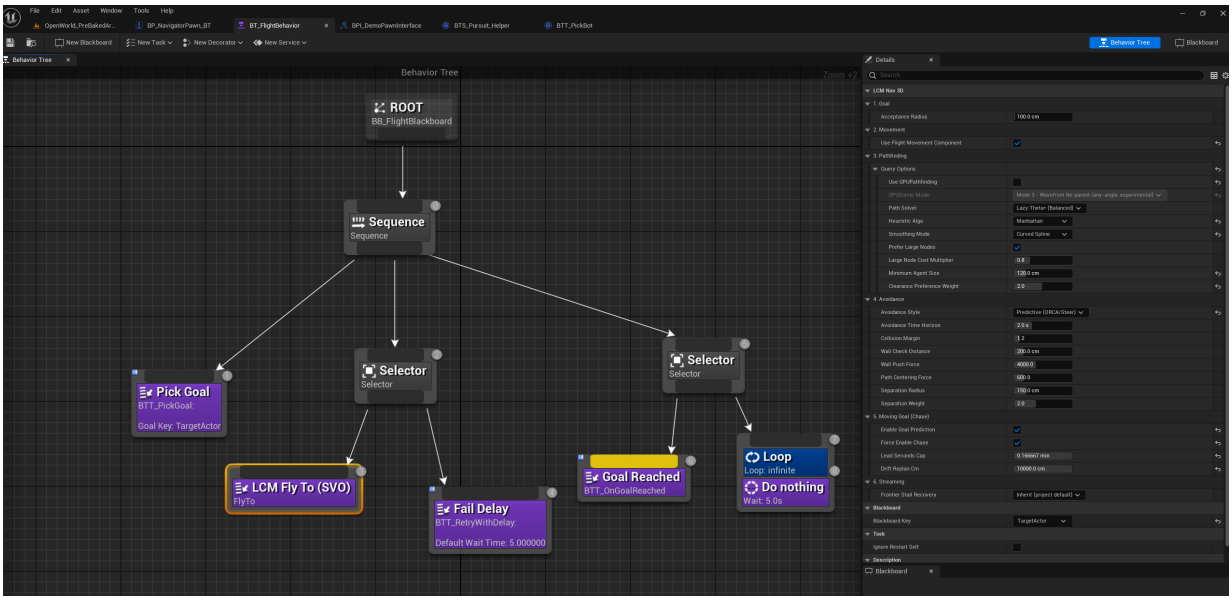


Figure 03.1 - Where `LCM Fly To (SVO)` sits. This is the shipped `BT_FlightBehavior` demo, one step past what you have just built: the same `Fly To` task, with a goal picker in front of it and a retry-with-delay branch beside it so a failed plan does not kill the tree. Yours needs only the highlighted node to start flying.

3.4.3 The FlyTo settings, in full

You will not need most of these on day one, but it is worth knowing the shape of the surface before you go looking for a specific knob.

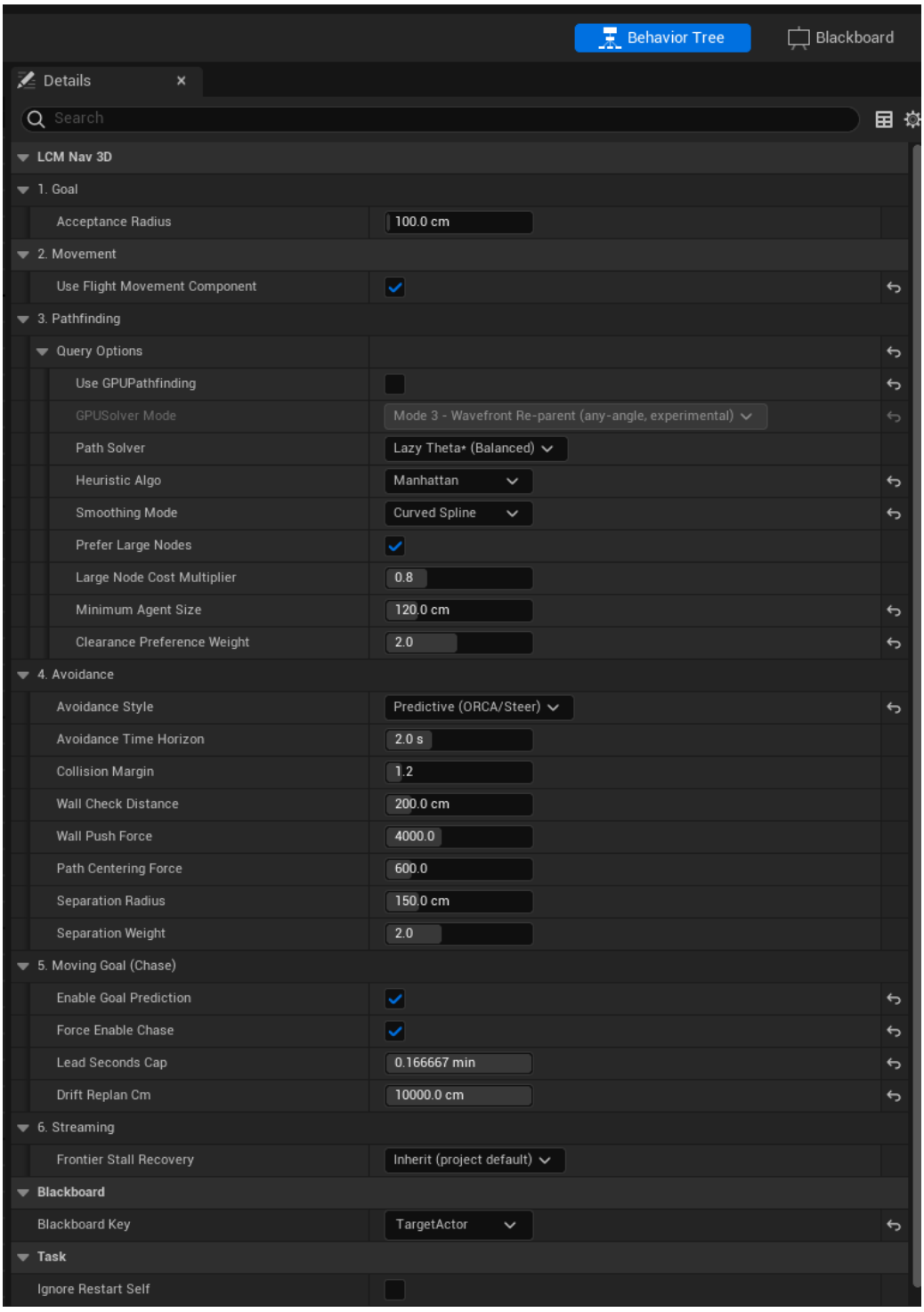


Figure 03.2 - The `LCM Fly To` task, all six sections. Read it as a pipeline: **1 Goal** is where you are going, **2 Movement** is what flies you there, **3 Pathfinding** is how the route is computed, **4 Avoidance** is how you get past other agents and walls, **5 Moving Goal** is only for chasing something that moves, and **6 Streaming** matters only in an infinite world.

Two fields decide the shape of everything below them:

- **`Use Flight Movement Component`** - ticked, the whole of section 2 collapses, because the movement component owns speed and dynamics instead of the task.
- **`Avoidance Style`** - set to `CBS`, the seven tuning fields under it hide; CBS is a planner-level solver and does not use them.

Sections you have not configured are not silently active. The panel hides any property the current configuration will ignore, so what you can see is what will run.

3.5 Step 4: The AI Controller

1. **Add → Blueprint Class → AIController**. Name it **`BP_MyFlyerAI`**.
2. Open it. On the **Event BeginPlay** node:
   - Add **Run Behavior Tree**, set **BTAsset** to `BT_MyFlyer`.
3. We need to put a goal in the blackboard. Still on BeginPlay, **after** Run Behavior Tree:
   - Add **Get Blackboard → Set Value as Vector**.
   - **Key Name:** `GoalLocation`
   - **Value:** for a first test, drag in a **Get Actor Of Class** node with **Actor Class** = **`TargetPoint`**, then **GetActorLocation** from it.

```
BeginPlay ─→ Run Behavior Tree (BT_MyFlyer)
          └→ Set Value as Vector (Key: GoalLocation,
                                  Value: GetActorOfClass(TargetPoint).GetActorLocation)
```

4. Compile and save.
5. Go back to **`BP_MyFlyer`** → Class Defaults → set **AI Controller Class** to `BP_MyFlyerAI`.

3.6 Step 5: Fly

1. Drag a **TargetPoint** into your level, somewhere your flyer has to climb or route around geometry to reach. Put it up high: that's the whole point.
2. Drag **`BP_MyFlyer`** into the level, somewhere with open air around it.
3. Press **Play**.

Your pawn should take off and fly to the target point, routing around geometry in full 3D.

3.7 If it doesn't move

Work down this list in order. These are the actual causes, in the order they occur.

Symptom	Cause	Fix
Output Log: **"No SVO Manager found in the level!"**	No manager actor	Step 1
Nothing happens, no log errors	AI never possessed the pawn	`Auto Possess AI` = *Placed in World or Spawned*, and `AI Controller Class` is set
Task fails instantly	Goal is outside the navigation volume	Increase `World Extent`, or move the target inside it
Task fails instantly	Goal (or start) is **inside** geometry	Move them into open air
Agent flies but ignores walls	Draw Debug Volumes shows no detail	`Min Voxel Size` too large
Agent refuses an obvious doorway	Opening sealed by padding	Lower `Clearance Padding` (see Tutorial 02 §5)
Agent jitters or sticks	Physics fighting navigation	Set mesh collision to No Collision for the test

Full list: Troubleshooting.

3.8 Step 6: Make it look good

Defaults are functional, not beautiful. Two changes transform the motion:

On the `LCM Fly To (SVO)` task:

Field	Try	Effect
`Flight Style`	**Physics (Drones/Ships)**	Momentum and inertia instead of instant direction changes. *(This is already the default.)*
`Flight Style`	**Linear (Biological/Magic)**	Instant, weightless. Good for spirits, insects, magic
`Banking Amount`	`4.0`	Rolls into turns like an aircraft
`Max Banking Angle`	`45`	How far it's allowed to roll
`Deceleration Distance`	`800`	Starts slowing this far out instead of stopping dead
`Max Acceleration`	`2000`	Lower = heavier, more sluggish feel

Path shape: a raw path is blocky. In Tutorial 02 we met the smoothing modes; **Curved Spline** is what makes flight look natural rather than like a series of ruler lines.

3.9 Step 7: Several agents at once

Duplicate `BP_MyFlyer` a few times and press Play. They'll fly the same route and crowd each other. Two settings fix that, both on the Fly To task:

Field	Default	Purpose
`Separation Radius`	`150`	How close before they push apart
`Separation Weight`	`2.0`	How hard they push

And **`Avoidance Style`**:

Style	Behaviour
None (Ghost/Ignore)	They fly through each other
Reactive (Boids/Push)	Push apart on contact: cheap, and the default
Predictive (ORCA/Steer)	Anticipate and steer around *before* colliding: smoother, costlier
Context-Based Steering (SVO)	Geometry-aware steering

For a handful of agents, **Reactive** is fine. For dozens sharing a corridor, **Predictive** looks markedly better.

> Past a few hundred agents, stop adding pawns and move to Tutorial 06: Mass Entity.

3.10 Step 8: Chasing a moving target

To chase something instead of flying to a fixed point:

1. Change the blackboard key `GoalLocation` to type **Object**, with **Base Class** = **`Actor`**.
2. Set it to the actor you want chased.
3. On the Fly To task, tick **`Enable Goal Prediction`**.

With prediction on, the agent aims where the target *will be* rather than where it is, so it intercepts instead of trailing. `Lead Seconds Cap` (default `2.0`) limits how far ahead it will extrapolate, and `Drift Replan Cm` (default `400`) controls how far the target must move before the path is recomputed.

3.11 Recap

- **One manager**, finite world, sensible `World Extent`.
- **Pawn + Floating Pawn Movement + Auto Possess AI**.
- **Blackboard key → `LCM Fly To (SVO)` → done.**
- **`Draw Debug Volumes` is your best debugging tool.**
- Tune feel with `Flight Style`, banking and deceleration; tune crowds with avoidance and separation.

Next: Tutorial 04: The Same Agent in StateTree, or jump to Tutorial 05: Dynamic Obstacles to make the world move.

CHAPTER 4

Tutorial 04: The Same Agent in StateTree

TL;DR: Identical flight behaviour to Tutorial 03, authored in StateTree instead of a Behavior Tree. Swap the AIController's `RunBehaviorTree` for a **`StateTreeAIComponent`**, and bind the **Fly To** task's `TargetLocation` instead of a blackboard key.

Time: ~15 minutes. **Do Tutorial 03 first.** This chapter only covers the differences.

4.1 Should you use StateTree or Behavior Tree?

Both are fully supported and produce the same flight. Pick on authoring preference, not capability.

	Behavior Tree	StateTree
Goal comes from	A **blackboard key**	**Bound instance data** on the task
Best at	Reactive priority ordering	Explicit states and transitions
Maturity	Stable	Epic's StateTree plugin is **experimental** on 5.2, stable from 5.3

> **The 5.2 caveat.** The plugin is flagged beta on 5.2 *only* because Epic's StateTree plugin is experimental there. If you're shipping on 5.2 and want the most conservative option, use the Behavior Tree path. On 5.3+ this distinction disappears.

4.2 Step 1: The pawn

Exactly as Tutorial 03 Step 2: a Pawn with **Floating Pawn Movement**, a visible mesh, and **Auto Possess AI = Placed in World or Spawned**.

Nothing StateTree-specific here.

4.3 Step 2: Create the StateTree asset

1. **Content Browser → Add → Artificial Intelligence → StateTree**.
2. When it asks for a **schema**, choose **StateTree AI Component Schema**. This matters: the AI schema is what exposes the AI context (controller and pawn) the Fly To task needs.
3. Name it **`ST_MyFlyer`**.

4.4 Step 3: Add the Fly To task

1. Open `ST_MyFlyer`. You'll have a **Root** state.
2. Add a child state and name it **`FlyToGoal`**.
3. With that state selected, in the **Tasks** section click **+** and choose **Fly To** (listed under the **LCM Nav3D** category).
4. Set the task's properties:

Field	Value	Notes
Target Location	your goal	See binding below
Acceptance Radius	`100`	
Flight Speed	`600`	Match your Floating Pawn Movement `Max Speed`

4.4.1 Binding the goal

This is *the* structural difference from Behavior Trees. There's no blackboard; you bind the task's `Target Location` to a value.

The simplest approach is a **StateTree parameter**:

1. Select the **Root** state → in **Parameters**, add a parameter: **Name `GoalLocation`, Type `Vector`**.
2. Select the **Fly To** task → click the bind icon next to **Target Location** → choose **`GoalLocation`**.

Now anything that sets that parameter drives the agent.

> **Chasing instead.** The task also has a **`Target Actor`** field. Set or bind that and the agent chases the actor, using the same prediction logic as the BT version. Note the ST chase path requires `Target Actor` to be non-null. If you want a fixed point, use `Target Location`.

4.4.2 Why `Fly To` has its own `Target Actor` field

This trips up everyone arriving from Behavior Trees, so it is worth stating plainly: **that field is not an alternative to `Find Actor By Tag` - it is what `Find Actor By Tag` plugs into.**

A Behavior Tree has a blackboard, so a BT task can name a key and read whatever wrote it. StateTree has no blackboard. Data moves by **property binding** from one node's output to another node's input, which means the *consuming* node must declare a field to receive the value. Delete `Target Actor` and there is nothing left to bind to.

This is the engine's own convention, not a plugin invention. Epic's `Move To` task carries exactly the same pair - a `Destination` vector plus an optional `Target Actor`, with the actor winning when both are set.

`Fly To` needs the live actor rather than a resolved position because it re-samples the target's location *and velocity* every tick to fly a lead intercept. A one-shot position would send the agent to where the target was when the state began.

4.4.3 Resolve the actor with the evaluator, not the task

Two nodes resolve a tag. Pick by whether you want a snapshot or a live reference:

	Find Actor By Tag (task)	**Find Actor By Tag** (evaluator)
Resolves	Once, on state entry	Continuously, on `Refresh Interval`
`Found Actor Location`	Frozen at entry	Refreshed **every tick**
Target destroyed / respawns	Stays stale until re-entry	Re-resolves immediately
Costs a state	Yes - unless `Stay Running After Resolve` lets it share one	No
Visible to	Nodes ordered after it	The whole subtree of the state it sits on
Can fail a state	Yes (returns `Failed`)	No - gate on its `b Found Actor` output

Prefer the evaluator for goal sources. Put it on the parent state, set `Tag` to `Player`, and bind `Fly To`'s `Target Actor` to its `Found Actor`. One resolver then feeds `Fly To`, `Look At`, conditions and transitions at once, and it stays correct when the target dies or streams in late.

Only the *search* is throttled by `Refresh Interval` (0.5 s by default); the location output refreshes every tick off the cached pointer, so a moving target is never stale. Reach for the task form when a frozen-on-entry snapshot is the point - picking a patrol anchor and committing to it - or when you specifically want the state to *fail* if nothing carries the tag.

> **Roaming between tagged goals.** The task form also has **`Pick Random Match`**: tag every goal actor in the level (say, `Goal`), put the task ahead of **Fly To** in one state, and tick **`Stay Running After Resolve`** - this switch is what lets the two tasks share a state. Off, the resolve *finishes on entry*, and a task that finishes finishes its **whole state**: Fly To is entered but never ticked, and the agent stands still. Bind Fly To's `Target Location` to `Found Actor Location`, then add an **On State Completed** transition back into the same state - as a **Go to State** transition targeting the state itself, *not* **Tree Succeeded**, which would end the whole run after the first leg. Each re-entry draws a fresh random goal - `Avoid Repeat Pick` (on by default) guarantees it is never the goal the agent is already sitting on - and every agent runs its own tree instance, so a crowd sharing one StateTree asset spreads across the goal set by itself. This is the tagged-actor twin of the **Pick Random Goal** task, which does the same loop with random *points* instead of placed actors and carries the same `Stay Running After Resolve` switch for the same reason.

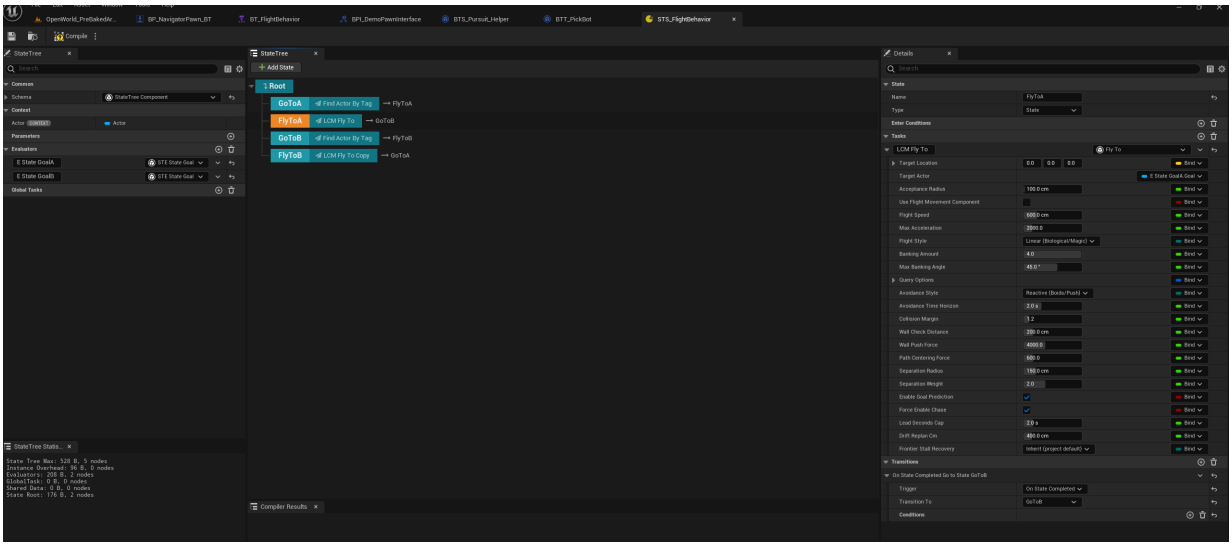


Figure 04.1 - A two-goal patrol, the StateTree way. This is the shipped `STS_FlightBehavior`. Each `FlyTo*` state runs one **LCM Fly To** task and transitions **On State Completed** to the next, so the patrol loop is expressed as states and transitions rather than as a Behavior Tree Loop decorator. The `Find Actor By Tag` states in between are what fill in each goal at runtime.

Note the **Evaluators** in the left panel - `E State GoalA` and `E State GoalB`. That is the StateTree replacement for a blackboard: an evaluator produces a value, and the task binds to it.

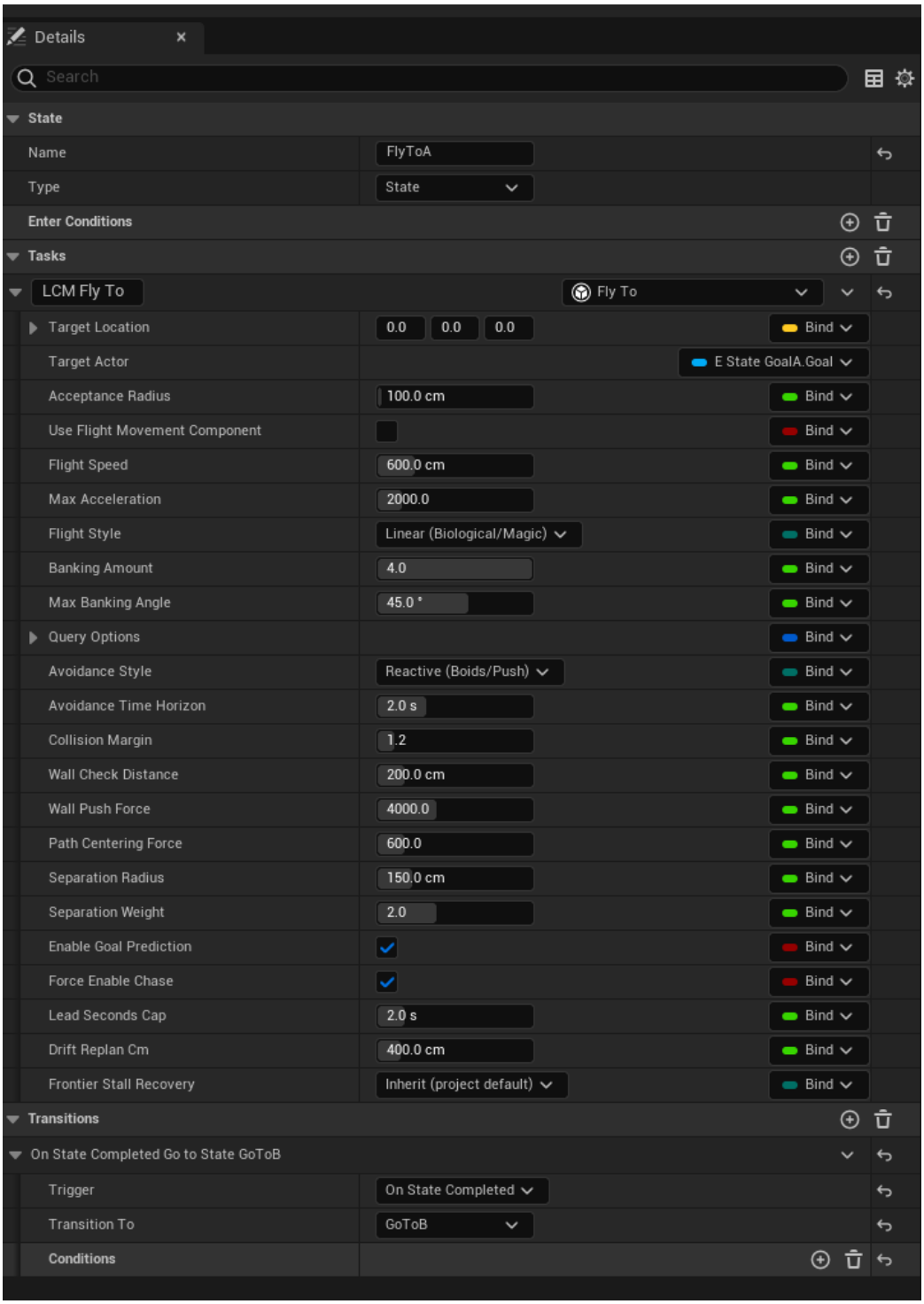


Figure 04.2 - The same task, in StateTree form. The properties are the ones you already met in Tutorial 03 - the difference is the **Bind** column down the right-hand edge. In StateTree *every* field can be driven by an evaluator, a parameter or another state's output, not just the goal.

In this shot `Target Actor` is bound to `E State GoalA.Goal` while everything else holds a literal, which is the normal arrangement: bind what changes, type what does not.

> **`Frontier Stall Recovery` reads `Inherit (project default)` here, and the task cannot tell you what that resolves to.** A StateTree asset is not owned by a level, so it does not know which manager it will run under. The manager's **7. Runtime State ▸ Frontier Stall Recovery (auto)** row is the one place that answer exists.

4.5 Step 4: The AI Controller

1. Create an **AIController** Blueprint, **`BP_MyFlyerAI_ST`**.
2. Add a **StateTree AI Component**.
3. Select it and set **State Tree** to `ST_MyFlyer`.
4. Set the goal parameter on BeginPlay. On the component, use **Set Parameter** (or set the value before the tree starts) with:
   - **Name:** `GoalLocation`
   - **Value:** e.g. `GetActorOfClass(TargetPoint) → GetActorLocation`
5. On **`BP_MyFlyer`** → Class Defaults → **AI Controller Class** = `BP_MyFlyerAI_ST`.

> Unlike the BT path there is no `Run Behavior Tree` call: the `StateTreeAIComponent` starts the tree itself.

4.6 Step 5: Fly

Place a **TargetPoint** and your pawn, press **Play**. Same result as Tutorial 03: the pawn routes through 3D space to the goal.

4.7 The rest of the StateTree surface

The plugin ships a full StateTree task set, all under the **LCM Nav3D** category:

Task	Does
Fly To	The flagship: 3D navigation to a point or actor
Patrol	Cycle through a set of points
Orbit Target	Circle a target at a radius
Land	Descend and touch down
Look At / Turn In Place	Orientation without translation
Find Cover	Move to tactical cover
Evade CBS	Avoidance-driven evasion
Follow Flow Field	Follow a shared crowd steering field
Chunk Prewarm / Streaming Aware	Infinite-world streaming helpers
Find Actor By Tag / Pick Random Goal / Wait	Utility
Use Smart Object	Smart Object interaction

Plus conditions (`Path Exists`, `Is At Location`, `Chunk Loaded`, `Has Line Of Sight`, `Is In Cover`, `Health Below`, `Path Length Below`, `Threat Visible Via Perception`) and evaluators (`Path Status`, `Path Progress`, `Navigation Load`, `Threat Field`, `Squad`, `Memory`, `Perception Relay`, `Find Actor By Tag`).

> **Reading a node's outputs.** Fields a node *produces* are grouped under an **Output** section and drawn as read-only `OUT` pins - those are the ones you bind *from*. Everything else is a parameter you either type a value into or bind *to*.

Every one is listed with its full property set in the API Reference.

4.8 Behaviour differences worth knowing

The BT and ST Fly To tasks are built for parity, but a few things genuinely differ:

	Behavior Tree	StateTree
Goal source	Blackboard key (Vector **or** Actor)	`Target Location` **or** `Target Actor` instance data
Chase	Works with a Vector or Actor key	Requires a non-null `Target Actor`
Arrival reported	Against the pre-move location that frame	Against the post-move location

The arrival difference is a sub-frame detail: it only matters if you're comparing the two side-by-side with a very small `Acceptance Radius`.

4.9 Troubleshooting

Symptom	Cause
Fly To isn't in the task list	Asset wasn't created with the **StateTree AI Component Schema**
Task fails immediately	No `LcmNavigationManagerSV0` in the level, or the goal is outside/inside geometry
Agent never starts	`StateTreeAIComponent` has no **State Tree** assigned, or `Auto Possess AI` is off
Goal is always (0,0,0)	The parameter isn't bound, or is set after the tree already ran

More: Troubleshooting.

Next: Tutorial 05: Dynamic Obstacles. Make the world move and watch agents reroute.

CHAPTER 5

Tutorial 05: Dynamic Obstacles

TL;DR: Add an **`LCM Nav3D Dynamic Obstacle`** component to any actor that moves. It restamps the octree as the actor moves and agents reroute around it: closing doors, rising platforms, patrolling hazards, destructible cover.

Time: ~15 minutes. Assumes Tutorial 03.

5.1 Static vs dynamic

When the navigation volume is built, all static geometry is baked in. That bake is what makes queries fast, but it's a snapshot.

- **Never moves?** Do nothing. It's already in the octree.
- **Moves?** It needs a **Dynamic Obstacle** component, or the octree will keep believing the space is empty and agents will fly straight through it.

> **The common mistake:** moving a static-mesh actor at runtime *without* the component. Nothing errors: agents ignore it, because as far as the navigation data is concerned it never moved.

5.2 Step 1: Make a moving obstacle

1. Create a Blueprint Actor, **`BP_MovingWall`**.
2. Add a **Static Mesh** component: a Cube, scaled to something that genuinely blocks a route (say 400 × 100 × 400).
3. Make sure the mesh **has collision enabled**. The voxelizer finds obstacles via collision, so a mesh set to *No Collision* is invisible to navigation.

5.3 Step 2: Add the Dynamic Obstacle component

1. In `BP_MovingWall`, **Add Component → search "LCM Nav3D Dynamic Obstacle"**.
2. That's the whole setup. It auto-registers with the navigation manager at `BeginPlay` and tracks its owner from then on.

5.3.1 The settings

Field	Default	What it does
Update Distance Threshold	`50` cm	How far the actor must move before the octree is restamped. Larger = cheaper, but navigation lags reality.
Tracked Components	*(empty)*	Empty means **the whole actor**. Pick specific components to track only part of it.

`Tracked Components` is worth knowing about. On an actor where only one part obstructs (a gate inside a decorative frame, a swinging arm on a fixed base), pick just that component. Only its movement restamps the octree, and only its bounds block agents.

You can also manage this at runtime from Blueprint with **`Add Tracked Component`** and **`Remove Tracked Component`**.

5.4 Step 3: Move it

Give it motion. The simplest version, in the Event Graph:

```
Event Tick ─→ SetActorLocation( StartLocation + (0, sin(GameTime) * 700, 0) )
```

Or use a Timeline, a matinee/sequencer track, physics; it doesn't matter. The component watches the transform, however it changes.

5.5 Step 4: See it work

1. Place `BP_MovingWall` so it periodically **seals the only route** between your flyer and its goal.
2. Press **Play**.

The agent should path around the wall, and when the wall closes the route, re-plan and take a different way.

To watch it happen: tick **`Draw Debug Volumes`** on the navigation manager. You'll see the voxels under the wall flip to blocked as it moves. That's the restamp.

5.6 How agents notice

Agents re-plan automatically. The relevant setting is on the **navigation manager**:

Field	Default	Meaning
4. Performance → `Finite Obstacle Update Interval`	`0.1 s`	How often obstacle changes are folded into the finite-world octree

Lower = more reactive, more CPU. `0.1` is a good balance. For fast-moving hazards where agents must react instantly, try `0.05`.

5.7 Ready-made examples

These demo maps show this working. Open any of them and read the setup:

Map	Shows
`Demo/BehaviourTree/PreBaked_FiniteMaps/DynObs_BT`	Obstacles sealing routes in a finite world
`Demo/BehaviourTree/InfniteMaps/DynObs_BT`	The same rig in a streaming infinite world
`Demo/BehaviourTree/InfniteMaps/DynObsTunnel_BT`	Movers sweeping a sealed tunnel: the wall-integrity test
`Demo/StateTree/InfniteMaps/DynObs_ST` · `DynObsTunnel_ST`	The StateTree equivalents
`Demo/MassEntity/PreBaked_FiniteMaps/DynObs_Mass`	Dynamic obstacles against a Mass crowd

Map	Shows
`Demo/MassEntity/InfniteMaps/DynObs_Tunnel_MassEntity`	A crowd flowing through a tunnel as movers sweep it

The obstacles in these maps are driven along splines by the bundled **`BP_SplineMove`** Blueprint (`Content/Blueprints/`), so you can see a complete working rig rather than assembling one. For a simpler ping-pong mover the plugin also ships the **`LCM Nav3D Obstacle Mover`** component, below.

5.8 The bundled Obstacle Mover

The plugin ships a small helper so the demo maps move without Blueprint wiring, and it's useful in your own prototypes.

`LCM Nav3D Obstacle Mover` ping-pongs its owner between its start pose and an offset.

Field	Default	Meaning
Seal Offset	`(400, 0, 0)`	Vector from the open pose to the sealed pose
Period	`4.0` s	Time for a full open → sealed → open cycle
Phase	`0.0` s	Offset so several obstacles don't move in lockstep

Pair it with a Dynamic Obstacle component on the same actor and you have a moving obstacle with zero scripting. It moves the actor kinematically, which is exactly what the obstacle tracker expects.

> There's a companion **`LCM Nav3D Agent Driver`** used by the demo maps to loop a pawn between two tagged points (`LcmGoalA` / `LcmGoalB`) using the real async navigation stack, handy for testing a level without authoring a full BT.

5.9 Performance notes

Dynamic obstacles are not free: each restamp re-voxelizes a region.

Keep it cheap:

- Prefer **fewer, larger** dynamic obstacles over many small ones.
- Raise **`Update Distance Threshold`** for things that move constantly but don't need precision.
- Use **`Tracked Components`** so only the genuinely blocking part is considered.
- If something moves but can never block a route (debris, VFX props, decoration), **don't give it the component at all**.

Don't put a Dynamic Obstacle component on your navigating agents. Agents avoid each other through the avoidance system (Tutorial 03 Step 7); making them obstacles too means each one is constantly re-voxelizing the world around itself.

5.10 Troubleshooting

Symptom	Cause
Agents fly straight through the obstacle	No Dynamic Obstacle component, or the mesh has collision disabled
Reaction is sluggish	`Update Distance Threshold` too high, or `Finite Obstacle Update Interval` too long

Symptom	Cause
Frame rate drops when things move	Too many dynamic obstacles, or they're too finely voxelized: raise `Min Voxel Size` or reduce obstacle count
Agent gets trapped when a route seals	Expected if there's genuinely no other route. Give the level a second path, or handle the failure in your BT/StateTree

More: Troubleshooting.

5.11 Where to go next

You now have the whole core loop: a navigation volume, agents that fly through it, and a world that can change underneath them. From here, pick what your project needs:

If you need...	Go to
Thousands of agents	Tutorial 06: Mass Entity
AI that picks tactical positions	Tutorial 07: Tactical EQS
Sight and awareness	Tutorial 08: Perception
Ground and flying AI together	Tutorial 09: Hybrid with Recast
Custom traversal or cost zones	Tutorial 10: Nav Links & Area Modifiers
To inspect it live in-game	Tutorial 11: Gameplay Debugger
To tune every setting	Settings Reference

Part II

Going Further

CHAPTER 6

Tutorial 06 - Mass Entity crowds (10K+ agents)

Add one trait to a Mass Entity Config and a crowd navigates in full 3D. No per-agent controller, no tick function, no code.

Time: ~20 minutes. **You'll end with:** thousands of agents flying to a goal.

> A `UActor` AI agent costs roughly 50 µs/tick once it has a controller, movement component and behaviour tree - a few hundred saturate the game thread. Mass represents agents as ECS fragments instead, so the same work runs in one processor sweep.

6.1 Prerequisites

- The `MassEntity` **and** `MassGameplay` plugins enabled. LCM Nav3D enables both; if you stripped them, re-enable both or the module fails to load.
- An `LcmNavigationManagerSV0` in the level with navigation built.
- Epic's "MassEntity Quick Start" if you have never used Mass - this tutorial only adds a trait on top of the standard flow.

6.2 Step 1: Create the agent config

Add → Miscellaneous → Data Asset → **`MassEntityConfigAsset`**. Name it `DA_MyCrowd`.

Open it and add three traits - one from each category:

Trait	Category	Set
LCM Nav3D Agent	Movement - ***exactly one***	`Nav Mode`, `Agent Radius`, `Max Speed`
LCM Nav3D Goal	Goal	`Goal Mode`, and the location or tag
LCM Nav3D ISM Render	Rendering	`Mesh`

6.2.1 Choosing a Nav Mode

Nav Mode	Use it for	Cost
Path Following	Agents with *different* goals, precise routes, infinite worlds	One path per agent
Flow Field	A crowd sharing *one* goal - the 10K-scale option	One field, shared by everyone

> ⚠ **Add a rendering trait or your agents are invisible.** They spawn and navigate correctly, they just draw nothing - the navigation trait deliberately owns navigation only. This is the most common first-run surprise.

6.2.2 Movement settings worth knowing

Setting	Default	What it does
`Max Turn Rate Deg`	`360`	The knob that turns square corners into arcs. `0` = instant snap
`Max Acceleration`	`2000`	cm/s². With turn rate, this is what makes movement look weighted
`Flight Style`	`Physics`	`Physics` (inertia) vs `Linear` (magic / biological, instant)
`Avoidance Style`	`Reactive`	`Reactive` (boids) / `Predictive` (ORCA, smooth mutual passing) / `None`
`Lane Bias`	`0.6`	Spreads a crowd across a corridor instead of balling up at turns

Defaults give natural movement out of the box.

6.3 Step 2: Spawn them

1. Place a **`MassSpawner`** actor in the level.
2. In **Entity Types**, reference `DA_MyCrowd`.
3. Set **Spawn Count** - start at 100 while you wire it up.
4. Give it a spawn-distribution asset so agents start at distinct positions **inside your navigation volume**.

6.4 Step 3: Give them a goal

The **LCM Nav3D Goal** trait does this with no code:

`Goal Mode`	Set	Behaviour
Fixed Location	`Goal Location`	Everyone paths to that point
Tagged Actor	`Target Actor Tag` (e.g. `ApexGoal`)	Everyone paths to the tagged actor, re-pathing if it moves

`Repath Tolerance Cm` decides how far a moving target drifts before agents re-path; `Arrival Tolerance Cm` is where they stop.

Write the goal and flip the status; the processors own every transition from there.

```
auto& Req = EntityManager.GetFragmentDataChecked<FLcmMassPathRequestFragment>(Handle);
Req.Goal   = TargetPos;
Req.Status = ELcmMassPathStatus::Pending;
```

`Status` moves `Pending → InFlight → Complete` (or `Failed`) and is readable at any time, so your own logic can branch on it.

6.5 Step 4: Check it works

1. Press **Validate Entity Config** on the asset. It catches the trait mistakes that are silent at runtime and all present as "my agents don't move".
2. Press Play with `r.LcmNav.Mass.DebugDraw 1` - agents draw as status-coloured spheres (grey idle, yellow pending, green complete, red failed) with a line to their waypoint.
3. Scale up gradually: 100, then 1,000, then 10,000, watching frame time.

6.6 If nothing moves

Symptom	Cause	Fix
Nothing visible at all	No rendering trait	Add **LCM Nav3D ISM Render**
Spheres are red	No path to the goal	Goal is outside the navigation volume, or inside geometry
Spheres stay grey	No goal set	Add the **Goal** trait, or write the request fragment
Validation error on save	Two movement traits, or Goal on a Flow Field agent	See Mass Trait Reference
Crowd vanishes when you walk away	World Partition streamed the spawner out	See the World Partition section of the trait reference

6.7 Tuning

Kill-switches and global overrides. Production authoring is the trait, not the console.

CVar	Default	What it does
`r.LcmNav.Mass.MaxPathsPerTick`	`64`	Spreads a large spawn wave's path requests across frames
`r.LcmNav.Mass.Avoidance`	`1`	Master switch for inter-agent avoidance
`r.LcmNav.Mass.DebugDraw`	`0`	Status-coloured spheres and waypoint lines

6.8 Scope

Mass agents resolve both endpoints **within the agent's own chunk**. Keep crowd goals in the same chunk as the crowd, or use a Behaviour Tree agent for long cross-chunk routes.

6.9 Related

- Mass Trait Reference - every trait, the combination matrix, and the World Partition trap.
- Tutorial 04 - StateTree, the per-entity AI layer that pairs with Mass.

CHAPTER 7

Tutorial 07 - Authoring true-3D tactical EQS

TL;DR: LCM Nav3D ships EQS generators and tests that work in **true 3D** - candidate points anywhere in the navigable volume, scored by volumetric SVO line-of-sight, cover, threat-exposure, LCM Nav3D-path-distance and height. Author a query from these nodes exactly like stock EQS; the only differences are the `Apex:`-prefixed generators/tests and that "distance" means *LCM Nav3D path*, not Euclidean or navmesh.

7.1 The LCM Nav3D EQS node catalog

Generators (`UEnvQueryGenerator`, produce 3D points): | Node | Produces | |---|---| | **LCM Nav3D | Points In SVO Volume** | uniform 3D points in an AABB, filtered to unblocked SVO leaves | | **LCM Nav3D | Hemispherical Ring 3D** | shells around a target (perch / orbit / high-ground); *Upper Hemisphere Only* option | | **LCM Nav3D | Points Along Path** | points sampled along the LCM Nav3D path between two contexts | | **LCM Nav3D | Portal Proximity** | points at macro-graph portal centres near a context (chokepoints) |

Tests (`UEnvQueryTest`, score/filter): | Node | Axis | Notes | |---|---|---| | **LCM Nav3D | Line Of Sight 3D** | visible to context (bool) | volumetric SVO LOS, not 2.5D | | **LCM Nav3D | Reachability 3D** | an LCM Nav3D path exists (bool) | filter unreachable candidates | | **LCM Nav3D | Path Distance 3D** | LCM Nav3D path length (float) | NOT Euclidean; Context = Querier | | **LCM Nav3D | Threat Exposure** | summed SweepPenalty toward threats (float) | lower = safer | | **LCM Nav3D | Cover Score 3D** | fraction of threats occluded-from (float) | higher = better cover | | **LCM Nav3D | Height Advantage** | Z over context (float) | high ground |

Contexts (`UEnvQueryContext`): | Node | Returns | |---|---| | **LCM Nav3D | Known Threats** | actors tagged `ApexThreat` | | **LCM Nav3D | Threats From AI Perception** | the querier's perceived threats (Tutorial 08) |

7.2 Step 1 - Author a sniper-perch query

Create an EQS query `EQS_FindSniperPerch`:

1. **Generator: LCM Nav3D | Hemispherical Ring 3D** - `GenerateAround` = the threat context (Known Threats / Threats From AI Perception), `RadiusMin/Max` = your engagement band, `Upper Hemisphere Only` = true (perch above the threat).
2. **Test: LCM Nav3D | Line Of Sight 3D** (Context = threats) - *Score, prefer should-have-LOS* - a perch must see its target.
3. **Test: LCM Nav3D | Cover Score 3D** (Context = threats) - *Score*↑ - partial cover is good.
4. **Test: LCM Nav3D | Height Advantage** (Context = threats) - *Score*↑ - favour high ground.
5. **Test: LCM Nav3D | Reachability 3D** (Context = Querier) - *Filter* - the agent must be able to get there.

The winning item is a reachable, high, in-cover position that still has line of sight to the threat - a sniper perch, chosen in full 3D.

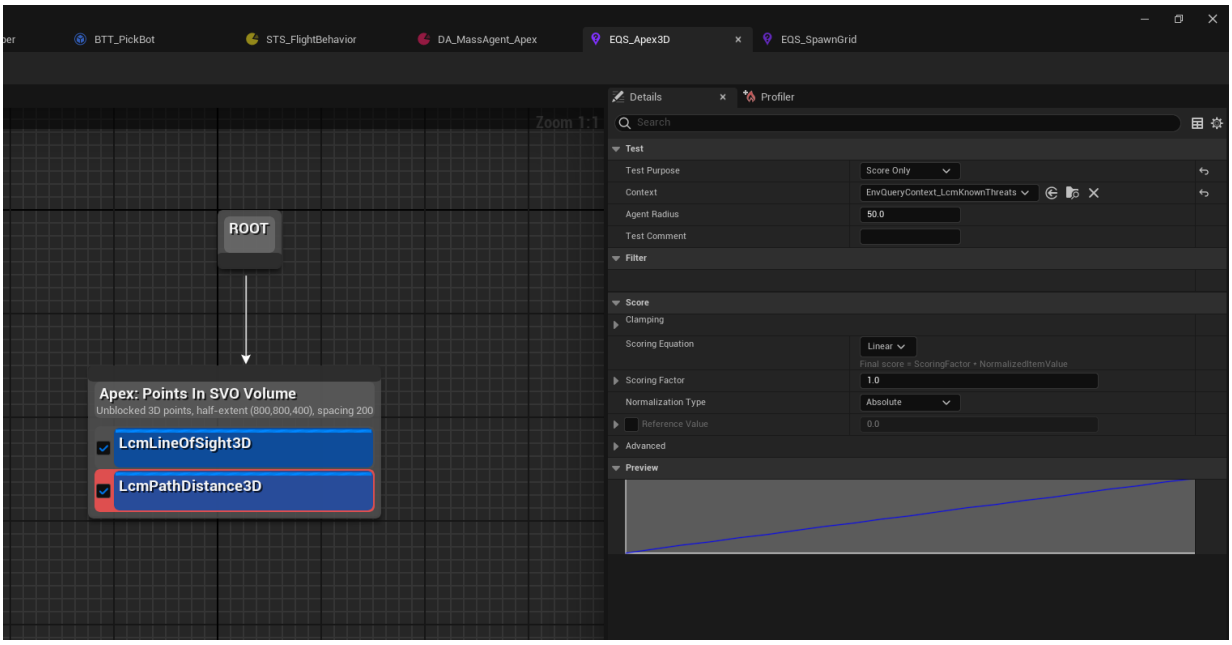


Figure 07.1 - The anatomy of a Nav3D query. One **generator** produces candidate points, then **tests** score or filter them, top to bottom. Here `Apex: Points In SVO Volume` is emitting unblocked 3D points across an 800×800×400 half-extent at 200 spacing - *unblocked* being the part a floor-plane generator cannot do, because it never had a third axis to be blocked in.

The two tests are colour-coded by role in the graph: a **filter** removes items, a **score** ranks them. The Details panel on the right belongs to the selected test:

- **`Test Purpose`** - `Score Only`, `Filter Only`, or both. This is the switch that decides whether a failing item is *penalised* or *deleted*.
- **`Context`** - what the test measures *against*. `EnvQueryContext_LcmKnownThreats` here, so line of sight is evaluated toward the threats the agent knows about.
- **`Agent Radius`** - how wide the thing that has to fit is. Leave this at your real agent size; a query that scores a perch a 50 cm drone can reach tells you nothing about a 200 cm gunship.
- **`Scoring Equation`** and the **Preview** graph - the curve mapping raw measurement to score. The preview updates live, so you can see a `Linear` ramp become a `Square` bias before you ever run the query.

7.3 Step 2 - The EQS Testing Pawn

Drop an **EQS Testing Pawn** in the level, set its `QueryTemplate` to `EQS_FindSniperPerch`, and place threats so the contexts resolve. The candidate items render as colour-graded balls; tweak generator/test parameters and watch the scores update live.

> UE's stock `EQSTestingPawn` is all you need - the LCM Nav3D nodes render through it normally.

7.4 Step 3 - Wire the result into a Behavior Tree

1. A `Run EQS Query` task/service runs `EQS_FindSniperPerch` and writes the best item to a blackboard `Vector` key.
2. A `MoveTo` (or the LCM Nav3D `FlyTo` BT task) moves the agent to that key.
3. Re-run on an interval (service) so the perch updates as threats move.

(StateTree: run the query from a task that stores the location, then transition to your LCM Nav3D move task - see Tutorial 05.)

7.5 The five shipped example queries

Recipes you can build now from the catalog above:

Query	Recipe
`EQS_FindSniperPerch`	Hemispherical Ring 3D + Cover Score 3D + LOS 3D + Height Advantage
`EQS_FindFlankingApproach`	Points In SVO Volume + Threat Exposure↓ + Path Distance 3D
`EQS_FindAmbushCorner`	Portal Proximity + LOS 3D (prefer blocked) + Reachability 3D

7.6 Related

- Tutorial 08 - perception-driven threat sources (the auto-populated threat context).
- Tutorial 05 - StateTree integration (the move side).
- `ULcmTacticalQueryLibrary` - the same primitives as Blueprint nodes (`Is Segment LOS Clear`, `Find Best Cover`, `Find Apex Path Sync`, ...).

CHAPTER 8

Tutorial 08 - Perception-driven tactical AI

TL;DR: Add an **LCM Nav3D AI Perception Bridge** next to your agent's `UAIPerceptionComponent` → its perceived threats auto-feed the **LCM Nav3D | Threats From AI Perception** EQS context → a true-3D LCM Nav3D EQS query (volumetric candidate points + 3D-LOS / cover / threat-exposure tests) picks a flanking or cover position → your BT/StateTree moves the agent there. Mass crowds get the same via the **LCM Nav3D Mass Perception Agent** trait, which writes a perceived-threats fragment.

8.1 Why this exists

Stock UE tactical AI flows through EQS, but the stock generators/tests are 2.5D (navmesh-projected). A flying or multi-level NPC can't reason about cover, line-of-sight, or flanking in true 3D with them. LCM Nav3D ships **true-3D EQS** - candidate points anywhere in the navigable volume, scored by **volumetric SVO** line-of-sight, cover, and threat exposure - and a **one-click bridge** so the threats those tests score against come straight from UE's own perception system.

8.2 Prerequisites

- LCMNav3D plugin enabled, an `ALcmNavigationManagerSVO` with a built SVO in the level.
- An AI agent with an `AAIController` and a `UAIPerceptionComponent` configured with an **AISense_Sight** config (the standard UE setup - LCM Nav3D doesn't change this).
- Threats set up so perception can sense them (registered stimuli source / pawns on a hostile team), per your normal UE perception configuration.

8.3 Part A - UActor agent (BT / StateTree)

8.3.1 Step 1 - Add the perception bridge

On your agent (the pawn or its controller - wherever the `UAIPerceptionComponent` lives), **Add Component** → **"LCM Nav3D AI Perception Bridge"** (`ULcmAIPerceptionBridge`).

- It auto-finds the owner's `UAIPerceptionComponent` on BeginPlay and subscribes to `OnTargetPerceptionUpdated`.
- `DecayWindowSeconds` (default `5`) - how long a lost-sight threat stays queryable before it fades. Tune to taste.
- No other wiring: successfully-sensed actors are tracked; lost ones decay out.

That's the entire perception-to-LCM Nav3D link.

8.3.2 Step 2 - Build the tactical EQS query

Create an EQS query (e.g. `EQS_FindFlankingApproach`) and assemble it from these nodes:

Generator (pick one): | Node | Use it for | |---|---| | **LCM Nav3D | Points In SVO Volume** | general 3D candidate cloud around the agent | | **LCM Nav3D | Hemispherical Ring 3D** | sniper-perch / high-ground / orbit candidates (set *Upper Hemisphere Only*) | | **LCM Nav3D | Points Along Path** | candidates strung along the LCM Nav3D path to a goal |

Tests (add several; set each one's **Context** to **LCM Nav3D | Threats From AI Perception** where it scores against threats): | Node | Purpose | Typical setup | |---|---|---| | **LCM Nav3D | Threat Exposure** | prefer low summed danger toward threats | Score, prefer **less** | | **LCM Nav3D | Cover Score 3D** | prefer positions occluded from threats | Score, prefer **greater** | | **LCM Nav3D | Line Of Sight 3D** | filter to (or away from) visible-to-threat | Filter | | **LCM Nav3D | Reachability 3D** | drop positions the agent can't path to | Filter (Context = Querier) | | **LCM Nav3D | Path Distance 3D** | prefer near / far by *LCM Nav3D* path length | Score (Context = Querier) | | **LCM Nav3D | Height Advantage** | favour high ground over threats | Score, prefer **greater** |

A solid "flank to cover" query: **Points In SVO Volume** → **Reachability 3D** (filter) → **Cover Score 3D** (score↑) → **Threat Exposure** (score↓) → **Path Distance 3D** (score↓, mild weight).

> The threats every threat-scoring test sees come from the bridge via **LCM Nav3D | Threats From AI Perception** - no manual tagging. (For non-perception workflows, **LCM Nav3D | Known Threats** returns actors tagged `ApexThreat` instead.)

8.3.3 Step 3 - Wire it into BT or StateTree, then verify

- **Behavior Tree:** add a `Run EQS Query` service/task that runs your query and writes the result `Vector/Actor` to the blackboard, then a `MoveTo` (or an LCM Nav3D `FlyTo`) to that key.
- **StateTree:** run the query from a task that stores the chosen location, then transition into your LCM Nav3D move task (see Tutorial 05).

Verify in PIE: the agent should pick positions that are reachable, in cover from, and low-exposure toward the threats it currently perceives - and should **stop reacting to a threat ~5 s after losing sight of it** (the decay window). Walk a threat out of the sight cone and watch the chosen position shift once it decays.

8.4 Part B - Mass crowd variant

For crowd-scale tactical perception, use the native LCM Nav3D Mass perception (Tutorial 06 covers Mass setup):

1. Add the **LCM Nav3D Mass Perception Agent** trait (`ULcmMassPerceptionTrait`) to your Mass entity config - set `FactionId`, `SightRadius`, `FOVDegrees`. Agents perceive different-faction agents within range + FOV + **true-3D SVO line of sight**.
2. The `ULcmMassPerceptionProcessor` writes each agent's closest visible hostile into its **`FLcmMassPerceivedThreatsFragment`** (`ClosestThreatPos`, `bHasThreat`, `VisibleCount`, `LastSeenSeconds`).
3. Read that fragment in your Mass logic (a StateTree task via `UMassStateTreeProcessor`, a steering processor, etc.) to drive evade / engage / regroup behaviour.

Mass agents consume the perceived-threats fragment **directly** - they don't issue EQS queries (there is no Mass-native EQS querier). EQS tactical queries are the UActor path (Part A). A hero-NPC-as-UActor querying with `Apex: Known Threats` can include crowd-relevant threats by tagging, giving a practical hybrid.

8.5 What you build by hand

The nodes above are all shipped and usable. What is **not** shipped is prebuilt query assets - you author your own from the generators and tests, which Tutorial 07 walks through end to end.

Four nodes need live flow-field or macro-graph state and are not available: the flow-field-directed generator, the flow-field-alignment test, the active-flow-field context, and the portal-proximity generator.

8.6 Related

- Tutorial 04 - StateTree, the move-task side of step 3.
- Tutorial 06 - Mass Entity, for the Part B agent setup.
- Tutorial 07 - Tactical EQS, authoring queries from these nodes.

- `ULcmTacticalQueryLibrary` - the Blueprint tactical primitives (`Is Segment LOS Clear`, `Find Best Cover`, …) the EQS tests wrap.

CHAPTER 9

Tutorial 09 - Hybrid LCM Nav3D + Recast crowd avoidance

TL;DR: Add `ULcmCrowdAvoidanceBridge` to your LCM Nav3D flying pawn → stock UE5 `UCrowdManager` Detour RVO sees the LCM Nav3D pawn and steers Recast crowd agents around it. The Z-band filter (auto-applied by the bridge) prevents false-collision avoidance between agents at different vertical bands.

9.1 Prerequisites

- The third-person template (or any project with a Character + AIController).
- LCMNav3D plugin enabled.
- An `ALcmNavigationManagerSVO` in the level.
- An `ALcmFlyingPawn` (or your own LCM Nav3D flying pawn).

9.2 Step 1 - Add the bridge to your LCM Nav3D pawn

In the editor, select your LCM Nav3D pawn → Add Component → search "LCM Crowd Avoidance Bridge".

Defaults are sensible:

- `bAutoRegister = true` - registers with `UCrowdManager` on BeginPlay automatically
- `AgentRadius = 50 cm` - typical pawn radius
- `AgentHalfHeight = 100 cm` - cylinder extent for Detour
- `MaxSpeed = 800 cm/s` - matches `ULcmNavMovementComponent` default

That's it for the LCM Nav3D side.

9.3 Step 2 - Add Recast crowd agents (existing UE5 pattern)

Your Recast ground crowd agents should already use `UCrowdFollowingComponent` instead of the default `UPathFollowingComponent`. This is the standard UE5 setup for any project using crowd avoidance - LCM Nav3D doesn't change this. If you're not already on `UCrowdFollowingComponent`:

```
// In your AAIController subclass constructor
GetCharacterMovement()->bUseAccelerationForPaths = true;
//. and override the AIController to use UCrowdFollowingComponent
```

Or in Blueprint: the AIController's `UCrowdFollowingComponent` is auto-registered as the agent.

9.4 Step 3 - Play

Hit Play. Your LCM Nav3D flying pawn and Recast ground crowd agents now avoid each other (per the Z-band filter rules below).

9.5 How the Z-band filter works

Unreal's `UCrowdManager` Detour RVO **assumes coplanar agents** - out of the box, a flying LCM Nav3D agent at Z=500 would be flagged as a "neighbour" by a Recast ground agent at Z=0, producing avoidance behaviour that looks like the ground agent dodging an obstacle 5 meters overhead.

The bridge fixes this at the **avoidance-group bitfield level**:

- Each agent gets an `AvoidanceGroup` bit derived from its world Z divided by `r.LcmNav.Crowd.ZBandHeightCm` (default 200 cm).
- Each agent's `GroupsToAvoid` is set to its own band + the ±1 adjacent bands (so vertically close agents still see each other).
- Cross-band agents (e.g. ground Z=0 and aerial Z=500 with 200 cm bands → bands 0 and 2) **ignore each other** via Detour's standard bitfield mask.

Same engineering work, leverages stock Detour rather than fighting it.

9.6 Tuning `r.LcmNav.Crowd.ZBandHeightCm`

Use case	Recommended value
Default (typical pawn-height scenes)	`200`
Low-ceiling interior (1.5m floors)	`150`
Vertical skyscraper (need fine-grained per-floor separation)	`100`
Open sky (most LCM Nav3D traffic is far above ground)	`400`
Disable filter (legacy Detour coplanar behaviour)	`0`

Set the CVar in PIE to live-tune, then commit to your project's `DefaultEngine.ini` once you're happy.

9.7 Stacked multi-floor scenarios

`UCrowdManager` is one global instance per world. For stacked-floor scenarios (e.g. LCM Nav3D agents at Z=200, 500, 800 sharing the same UCrowdManager), the Z-band filter handles it automatically - each pair of agents whose Z-distance is > 1 band-height ignores each other.

For **truly independent** crowds (each floor should be its own crowd zone), you'd need to spawn separate `UCrowdManager` instances per zone via project-side subclassing. This is out of scope for v1.

9.8 Performance notes

- Each registered LCM Nav3D bridge contributes ~1 `RegisterAgent` call at BeginPlay + ~7 virtual-call evaluations per Detour tick (Detour reads the ICrowdAgentInterface getters once per query).
- The Z-band filter eliminates ~99% of cross-band RVO computation cost - only agents within ±1 band of you contribute to your avoidance solve.
- Disabling the filter (`r.LcmNav.Crowd.ZBandHeightCm = 0`) restores legacy Detour coplanar behaviour for benchmarking.

9.9 Acceptance test (manual)

1. Spawn 5 Recast ground crowd agents in a circle at Z=0.
2. Spawn 1 LCM Nav3D flying pawn with `ULcmCrowdAvoidanceBridge` at Z=500.
3. Fly the LCM Nav3D pawn through the middle of the ground-agent circle.
4. **Expected**: ground agents continue their pathing without dodging the airborne pawn (Z-band separation > 200cm default).

5. Now fly the LCM Nav3D pawn down to Z=100 - ground agents dodge it.

If step 4 fails (ground agents dodge the airborne pawn), check `r.LcmNav.Crowd.ZBandHeightCm` is not 0.

9.10 What about the LCM Nav3D pawn's own avoidance?

Two paths, both work:

1. **LCM Nav3D CBS/ORCA** (default, the FlyTo task's built-in steering): the LCM Nav3D pawn avoids Recast crowd agents via the existing CBS context. Detour Crowd is purely the *publishing* surface to the rest of the world.
2. **Use UCrowdFollowingComponent on the LCM Nav3D pawn**: replace the LCM Nav3D pawn's AIController's `UPathFollowingComponent` with `UCrowdFollowingComponent`. The pawn becomes a full Detour Crowd participant - receives the suggested-velocity from Detour and applies it through its movement component. Combine with `ULcmNavMovementComponent` for the locomotion side.

Path 1 is the default; Path 2 is for projects that want a uniform crowd avoidance solver across LCM Nav3D and Recast agents.

9.11 Mass variant

The Mass-Entity equivalent (`UApexCrowdMassBridgeProcessor`) ships with T26 Mass Entity work - not yet available.

9.12 Related LCM Nav3D surfaces

- `LCM_BTTask_FlyTo` - already has CBS+ORCA built-in for LCM Nav3D-internal avoidance
- `LCM_BTTask_EvadeCBS` - pure-defensive evade using the same CBS context
- `ULcmNavMovementComponent` - LCM Nav3D-aware flying-pawn movement component

CHAPTER 10

Tutorial 10 - 3D NavLinks + volumetric NavArea modifiers

TL;DR: LCM Nav3D ships `ULcmNavLinkComponent` for vertical ladders/jump shafts/teleporters (3D-advantage over Recast's surface-projected links) and `ALcmNavAreaModifier` for volumetric cost overlays (storm clouds, water volumes, danger zones, preferred lanes). Both auto-register on BeginPlay; query via `ULcmNavAreaQueryLibrary` from BP or C++.

10.1 The 3D-advantage angle

Unreal's stock `UNavLinkCustomComponent` is fine for **flat** NavLinks projected onto the Recast navmesh. But:

- **Vertical ladders** spanning 3 floors → Recast needs the navmesh at every floor + the link projects to those poly centres
- **Drop-down jumps** from a rooftop to a balcony at Z=−500 → Recast clamps the endpoints to nearest navmesh polygons
- **Teleporters** between Z=200 and Z=2000 → completely outside Recast's representation

LCM Nav3D flying pawns can use **any two world-space points** for the link endpoints. The components below surface that capability so designers don't have to hand-roll a 3D-aware link system.

10.2 Step 1 - Drop a 3D vertical NavLink (ladder example)

1. In the editor, spawn an empty actor (or reuse `ANavLinkProxy`).
2. Add Component → search "LCM Nav Link (3D Vertical)".
3. Set on the component:
   - `LinkStart` = world position of the ladder bottom (e.g. `(0, 0, 0)`)
   - `LinkEnd` = world position of the ladder top (e.g. `(0, 0, 800)` - 8 m vertical)
   - `bBidirectional` = true (climb up or slide down)
   - `bAutoRegisterWithApex` = true (default)
   - `TraversalCostCm` = extra cm-cost of using the ladder (0 = free; e.g. `400` to make agents prefer a nearby ramp). Queryable in v1 (see Step 3); auto-routing is v2.

That's it. The link is now visible to:

- Stock Unreal nav-system tooling (it implements `INavLinkCustomInterface`)
- LCM Nav3D's extension registry (queryable via `ULcmNavAreaQueryLibrary`)

10.3 Step 2 - Drop a volumetric NavArea modifier (water example)

1. In the editor, spawn a `ALcmNavAreaModifier` actor.
2. Move + scale the box gizmo to cover your water volume.
3. Set:
   - `CostMultiplier` = `3.0` (agents prefer dry routes by 3× cost)
   - `AreaTag` = `"Water"` (optional designer label)
   - `bAutoRegisterWithApex` = true

Stack multiple modifiers in the same volume - costs **multiply**:

Volume	Cost multiplier
Water	3.0
+ storm cloud overlapping	× 5.0
Effective in overlap region	15.0

10.4 Step 3 - Query from BT / StateTree / Blueprint

```
// In a BT service tick, or anywhere with C++ access:
const float Cost = ULcmNavAreaQueryLibrary::GetCostMultiplierAt(AgentLocation);

// Or evaluate a planned path's accumulated area cost (multiplicative):
const float PathCost = ULcmNavAreaQueryLibrary::GetCostMultiplierAlongPath(Waypoints);

//.and the additive cm-cost of any priced NavLinks the route uses:
const float LinkCost = ULcmNavAreaQueryLibrary::GetNavLinkTraversalCostAlongPath(Waypoints);
// Combine however your AI weighs routes, e.g. compare two candidate paths and pick
// the one with the lower (Length * PathCost + LinkCost).

// Find the nearest ladder for "approach the ladder" patterns:
ULcmNavLinkComponent* Nearest =
    ULcmNavAreaQueryLibrary::FindNearestNavLink(AgentLocation, /*MaxDist*/ 500.0f);
if (Nearest)
{
    FVector LadderStart, LadderEnd;
    ENavLinkDirection::Type Dir;
    Nearest->GetLinkData(LadderStart, LadderEnd, Dir);
    // Have FlyTo task move to LadderStart, then trigger the link traversal.
}
```

All of these are Blueprint-callable via the standard "LCM Navigation | Bridge" category.

10.5 Solver integration (v1 vs v2)

v1 (this release):

- NavLinks register with the UE5 NavSystem → stock `UBTTask_MoveTo` workflows that involve smart-links work
- NavArea cost multipliers **and** NavLink `TraversalCostCm` are **queryable** (`GetCostMultiplierAlongPath` / `GetNavLinkTraversalCostAlongPath`) but the LCM Nav3D A* solver doesn't yet incorporate them into its own path-cost evaluation
- Buyer pattern: use the query library in a BT service to write the cost-along-path into a blackboard key, then a `LcmDecorator_PathLengthBelow`-style decorator gates on it

v2 (deferred):

- Cost multipliers plumbed into the LCM Nav3D A* leaf-cost evaluation → paths actually route around / through preferred lanes automatically
- NavLink edges added to the macro graph so the solver plans through them
- An additional cost channel under the same bounded-LSB parity discipline as the existing sweep-penalty plumbing

v1 is the placement + registry + query surface; v2 is the solver-side cost integration. The data plane is ready; the planner-side wiring lands when the Marketplace v3.0 ramp justifies the solver-touch budget.

10.6 Common patterns

10.6.1 Ladder + drop-down combo

Two `ULcmNavLinkComponents` on the same actor:

- Ladder: bidirectional, `LinkStart = (0, 0, 0)`, `LinkEnd = (0, 0, 800)`
- Drop-down: one-way, `LinkStart = (200, 0, 800)`, `LinkEnd = (200, 0, 0)`, `bBidirectional = false`

10.6.2 Storm cloud + ground danger

Two `ALcmNavAreaModifierS`:

- Storm cloud at Z=300.800, CostMultiplier 5.0, Tag "Storm"
- Danger zone at Z=0.150, CostMultiplier 10.0, Tag "Combat"

Agents flying at Z=200 see normal cost; flying at Z=400 see 5× cost; flying through ground combat at Z=50 see 10× cost.

10.6.3 Preferred lane

Single `ALcmNavAreaModifier` with `CostMultiplier = 0.5`. LCM Nav3D paths bias toward this volume when the v2 solver integration lands.

10.7 Acceptance test (manual)

1. Drop a `ULcmNavLinkComponent` with LinkStart at `(0,0,0)`, LinkEnd at `(0,0,500)`, and `TraversalCostCm = 400`.
2. In a Blueprint test actor: `Print String (ULcmNavAreaQueryLibrary::GetRegisteredLinkCount)` → should print `1`.
3. Move the test actor near `(0,0,0)`. Call `FindNearestNavLink(actor.location, 1000)` → returns your component.
4. `GetNavLinkTraversalCostAlongPath([(0,0,0),(0,0,500)])` → returns `400`. A path that stays away from the link (e.g. `[(5000,0,0),(6000,0,0)]`) → returns `0`.
5. Drop a `ALcmNavAreaModifier` covering `(0,0,250)` with CostMultiplier 5.0.
6. `GetCostMultiplierAt((0,0,250))` → returns `5.0`. At `(0,0,800)` → returns `1.0`.

10.8 Related LCM Nav3D-unique surfaces

- `ULcmTacticalQueryLibrary::GetSweepPenaltyAt` - §P9.3 dynamic-obstacle risk byte (orthogonal to NavArea modifiers; both apply)
- `LCM_BTDecorator_PathLengthBelow` - LCM Nav3D path-length predicate; complements cost-along-path
- `LCM_BTService_ThreatFieldSampler` - write SweepPenalty into BB; pair with cost-multiplier write for full risk-aware AI

CHAPTER 11

Tutorial 11 - LCMNav3D in the Gameplay Debugger

TL;DR: LCMNav3D ships an `LCMNav3D` category in the stock UE5 Gameplay Debugger. In PIE, press the standard Gameplay Debugger key (usually apostrophe `'`), select an agent → the category renders per-agent + plugin-wide state alongside Recast / BT / EQS / Perception. Zero project setup beyond the plugin being loaded.

11.1 What it shows

11.1.1 Plugin-wide block (always)

```
== LCMNav3D ==
  Manager: resolved
  Loaded chunks: 7
  NavLink registry: 3
  NavArea registry: 1
  NavSystem bridge: active
```

Field	Meaning
`Manager`	Green when `ALcmNavigationManagerSV0` is in the world; red if missing
`Loaded chunks`	`ActiveVirtualChunks.Num` for infinite worlds; 1 for finite
`NavLink registry`	Count of `ULcmNavLinkComponent` currently registered
`NavArea registry`	Count of `ALcmNavAreaModifier` currently registered
`NavSystem bridge`	Green when the T37 proxy is registered with UNavigationSystemV1

11.1.2 Per-agent block (when an actor is selected)

```
== Agent: BP_FlyingPawn_C_0 ==
  Pos: X=1234.5 Y=678.9 Z=500.0
  Speed: 624 cm/s
  Chunk: (0,0,0) full
  SweepPenalty: 0
  Perceived: 2 (closest: BP_Enemy_C_1)
```

Field	Meaning
`Pos`	Pawn world location
`Speed`	`Pawn->GetVelocity.Size` in cm/s
`Chunk`	`(IntVec3) resolution` - `full`, `coarse` (T12 stub), or `none` (streaming gap)
`SweepPenalty`	§P9.3 risk-field byte at the agent's location (0.255)
`Perceived`	Count of currently-perceived actors + closest threat name

11.2 How to toggle

1. In PIE, press the standard Gameplay Debugger key (usually apostrophe `'` - check `Project Settings → Engine → Gameplay Debugger` if you've rebound it).
2. The Gameplay Debugger HUD appears at the top of the viewport.
3. Look for the `LCMNav3D` category - it should be enabled by default (state set to `EnabledInGameAndSimulate` at module init).
4. Click an actor in the level to select it as the debug target - per-agent block populates.

If the category isn't showing:

- Confirm you're in PIE (Gameplay Debugger only runs in PIE/Standalone, not the editor itself).
- Confirm the plugin is loaded (check Output Log for `[LcmNavBridge]` messages on map load).
- Confirm you're not in a Shipping/Test build (the category compiles out when `WITH_GAMEPLAY_DEBUGGER=0`).

11.3 What survives PIE → editor → PIE reloads

- **CVars** (debug toggles, chase tuning, OTel NDJSON, etc.) - persist across PIE sessions, reset on editor restart.
- **Profile loader state** (`ULcmNavProfileSubsystem`) - re-applied on each PIE init if the profile path is in your project's startup.
- **Telemetry buffer** - flushed at module shutdown; survives PIE-stop but not editor-close.
- **NavLink + NavArea registry** - populated at each actor's `BeginPlay`; PIE-stop clears via `EndPlay`.
- **NavSystem bridge proxy** - spawned per-world; new PIE session = new proxy.

The Gameplay Debugger category itself is **always-on after module load** - it doesn't lose its registration across PIE sessions because the category lives in the GameplayDebugger module's category map, not in the world.

11.4 Adding new fields (v2 design hook)

To extend the category in your project subclass:

```
#if WITH_GAMEPLAY_DEBUGGER
class FProjectGameplayDebuggerCategory_Apex: public FGameplayDebuggerCategory_LcmNav3D
{
public:
    virtual void CollectData(APlayerController* PC, AActor* DebugActor) override
    {
        Super::CollectData(PC, DebugActor);
        // Gather project-specific state here
    }
    virtual void DrawData(APlayerController* PC, FGameplayDebuggerCanvasContext& Canvas) override
    {
        Super::DrawData(PC, Canvas);
        Canvas.Printf(TEXT("  ProjectField: %s"), *MyProjectState);
    }
};
#endif
```

Register your subclass instead of the stock one in your project's module startup.

11.5 Performance

- `CollectData` is called once per tick per active category. Cost ≈ 10 µs (one `GetActorOfClass` + a few accessors + a perception query when an actor is selected).
- `DrawData` runs only when the HUD is visible. Cost ≈ negligible (text rendering).
- Shipping builds: **zero cost** (entire category compiles out via `WITH_GAMEPLAY_DEBUGGER` gate).

11.6 Related LCM Nav3D debug surfaces

- `r.LcmNav.Debug.StateTreeOverlay` - on-screen pawn state DrawDebugString
- `r.LcmNav.Debug.CoverViz` - cover-spot DrawDebugSphere overlay
- `r.LcmNav.Debug.FlowFieldViz` - flow-field arrow grid
- `LcmNavStress.SpawnAgents <N>` - stress-spawn helper for scale testing

Part III

Behaviour Tree Stock Patterns

CHAPTER 12

BT Wait - use stock `UBTTask_Wait`

TL;DR: Unreal Engine ships `UBTTask_Wait`. Use it.

12.1 Pattern

1. In your BT graph: right-click → Add Task → "Wait".
2. Set `WaitTime` (seconds) + optional `RandomDeviation`.
3. Task returns `Succeeded` after the timer.

For a wait that depends on a blackboard-stored time, use `UBTTask_WaitBlackboardTime` instead.

12.2 Why we didn't ship `LCM_BTTask_Wait`

Bit-identical functionality. The LCM Nav3D StateTree variant `FLcmSTTask_Wait` is shipped because StateTree on 5.2 doesn't include a stock Wait task - but BT has one since 4.x. No additive value from a custom LCM Nav3D variant.

12.3 LCM Nav3D-specific wait variant

If you want to wait for a **specific LCM Nav3D condition** (e.g. "wait until the chunk at position X is loaded"), use the LCM Nav3D-unique:

- `LCM_BTTask_ChunkPrewarm` - waits for chunk readiness with a configurable timeout, proactively enqueues the chunk for generation

CHAPTER 13

BT IsAtLocation - use the stock UE5 decorator

TL;DR: Unreal Engine already ships `UBTDecorator_IsAtLocation`. Use it directly on LCM Nav3D pawns. No LCM Nav3D-specific variant exists or is needed.

13.1 Why we didn't ship `LCM_BTDecorator_IsAtLocation`

This audit found that `UBTDecorator_IsAtLocation` already does what an LCM Nav3D-side variant would do:

- Reads a `Vector` or `Object` blackboard key
- Computes distance from the pawn to the key
- Returns true if within `AcceptableRadius`
- Optional `bPathFindingBasedTest` reuses the active path-following result

This is the canonical "did the AI reach the location" predicate. Re-shipping it on the LCM Nav3D side would be redundant and force buyers to pick between two identical nodes.

13.2 Three-step usage on an LCM Nav3D pawn

1. **Drop the stock decorator on a Sequence / Selector node** in your BT graph. Right-click → Add Decorator → "Is At Location".
2. **Set the blackboard key.** Vector key (a goal position) or Object key (an AActor whose location is the goal - `bUseNavAgentGoalLocation` projects to nav-agent location). For LCM Nav3D flying pawns where you want the literal world position, use Vector.
3. **Set `AcceptableRadius`** to your "we count this as arrived" distance in cm. Typical: 50-150 cm.

That's it. The decorator works on every LCM Nav3D pawn that has an AIController - no LCM Nav3D-specific configuration needed.

13.3 When to use the path-finding variant

`UBTDecorator_IsAtLocation::bPathFindingBasedTest` makes the decorator consistent with whatever was used to reach the location - useful when the move was via `UBTTask_MoveTo` (Recast) so the "arrived" check matches.

For LCM Nav3D moves, this option will fall through to Recast even if your goal was inside an LCM Nav3D volume - leave it OFF and use straight geometric distance until **T37 NavSystem bridge** lands. After T37, the stock `bPathFindingBasedTest` will route through LCM Nav3D automatically when the goal is in an LCM Nav3D-managed region.

13.4 Related LCM Nav3D-unique decorators

For predicates that **don't** have a stock equivalent, see the LCM Nav3D-unique decorators we did ship:

- `LCM_BTDecorator_PathLengthBelow` - true-3D LCM Nav3D path-length check (stock `DoesPathExist` is binary + Recast)
- `LCM_BTDecorator_HasLineOfSight` - voxel-aware Lazy Theta* LOS (stock uses raycasts only)
- `LCM_BTDecorator_IsInCover` - composite SweepPenalty + LOS predicate
- `LCM_BTDecorator_ChunkLoaded` - SVO streaming probe
- `LCM_BTDecorator_ThreatVisibleViaPerception` - AIPerception + LCM Nav3D LOS composite

These are genuinely additive surfaces; "IsAtLocation" is not - use stock.

CHAPTER 14

BT LookAt - use stock `UBTTask_RotateToFaceBBEntry`

TL;DR: Unreal Engine ships `UBTTask_RotateToFaceBBEntry`. It covers LookAt + TurnInPlace use cases. Use it.

14.1 Pattern

1. In your BT graph: right-click → Add Task → "Rotate to Face BB Entry".
2. Configure:
   - `BlackboardKey`: a Vector key (target location) or Object key (target actor).
   - `Precision`: degrees of yaw tolerance to count as facing.
3. Task returns `Succeeded` when the agent's rotation is within `Precision` of the target direction.

The stock task uses the AIController's `SetFocus` / `ClearFocus` mechanism so the pawn's rotation continues smoothly even after the task ends (if you want that, use `UBTService_DefaultFocus` to keep focus while a parent node is active).

14.2 Why we didn't ship `LCM_BTTask_LookAt` or `LCM_BTTask_TurnInPlace`

Both planned variants do exactly what `UBTTask_RotateToFaceBBEntry` already does:

- `LookAt`: rotate-toward-target with interp speed → stock task has built-in smooth interp via the AIController focus system
- `TurnInPlace`: constant-rate rotation toward target → stock task supports configurable interp rate

The LCM Nav3D StateTree variants (`FLcmSTTask_LookAt`, `FLcmSTTask_TurnInPlace`) are shipped because StateTree on 5.2 has no stock rotation task - but BT has one since 4.x.

14.3 When to write a custom LCM Nav3D rotation task

The only case where stock doesn't work: you need **physics-integrated banking** matching the flying-pawn locomotion style. That's already integrated into `LCM_BTTask_FlyTo`'s rotation phase. If you want LookAt-without-translation but WITH banking, fold it into FlyTo with a tiny step size or use the StateTree variant which does have banking-aware rotation.

For >95% of "face the target" use cases, stock `UBTTask_RotateToFaceBBEntry` is the right answer.

CHAPTER 15

BT TurnInPlace - use stock `UBTTask_RotateToFaceBBEntry` + `UBTService_DefaultFocus`

TL;DR: Same stock task as LookAt. Combine with `DefaultFocus` service for "keep facing while doing other tasks" patterns.

15.1 Pattern: simple turn

Use `UBTTask_RotateToFaceBBEntry` exactly as in `BT_LookAt_StockPattern.md`. The task completes when the agent has rotated to face the BB key.

15.2 Pattern: continuous facing during other tasks

If you want the agent to **keep facing a target** while executing other tasks in a parent sequence:

1. Add `UBTService_DefaultFocus` to the parent **Selector** or **Sequence**.
2. Set the focus BB key (Vector or Object).
3. The AIController's focus stays set for the entire subtree - pawn faces the target via `SetFocus` while other tasks execute.

This is the canonical UE5 pattern for "shoot while strafing" combat AI. No LCM Nav3D-specific variant needed.

15.3 Why no `LCM_BTTask_TurnInPlace`

Stock `UBTTask_RotateToFaceBBEntry + UBTService_DefaultFocus` covers the entire use case. The LCM Nav3D StateTree `FLcmSTTask_TurnInPlace` exists because StateTree v0.1 doesn't have the equivalent stock task; on BT side it's redundant.

15.4 Constant-rate turn (the original spec)

If you specifically need **constant angular velocity** (not stock's interp-with-acceleration), wrap a small BP-only task: read pawn rotation, target rotation, apply `MaxRotationRate * DeltaTime` step toward target. ~10 BP nodes. Or use the StateTree variant which supports this directly.

For the typical "snap-look-at-threat" pattern that AAA combat AI uses, stock is the right answer.

CHAPTER 16

BT HealthBelow - use the stock blackboard + decorator pattern

TL;DR: Unreal Engine ships everything you need: a Float blackboard key + a Blackboard-Compare decorator. No LCM Nav3D variant.

16.1 Pattern

1. **In your Blackboard asset**: add a Float key `CurrentHealth`.
2. **In your AI controller / game code**: write `CurrentHealth` whenever damage is applied.
3. **In your BT graph**: add a `Blackboard` decorator to the "retreat" / "panic" sequence. Set:
   - Key: `CurrentHealth`
   - Notify Observer: On Value Change (so the decorator re-evaluates immediately on damage)
   - Operator: `Is Less Than`
   - Value: `30.0` (or whatever threshold)

That's the canonical UE5 way of doing "if health below X, do Y."

16.2 If you want a GameplayTag-based pattern

For projects using GAS where health states are tags (e.g. `Status.Bleeding`, `Status.LowHealth`), use the stock `UBTDecorator_CheckGameplayTagsOnActor`. Same semantics, tag-driven.

16.3 Why we didn't ship `LCM_BTDecorator_HealthBelow`

The stock blackboard-compare decorator covers the exact same functionality with one extra dropdown (the comparison operator) - which actually gives the buyer more flexibility (`> < <= >= ==`) than a fixed "below threshold" LCM Nav3D decorator. The audit classified this as a partial overlap; the buyer-experience answer is to lean on stock.

The LCM Nav3D StateTree variant `FLcmSTCondition_HealthBelow` is kept for ergonomic parity (named clearly in the StateTree node picker), but its underlying behaviour is also "float compare" - stock-equivalent.

16.4 Related LCM Nav3D-unique decorators

If your "should I retreat" gate isn't health-based but **threat-field-based**, the LCM Nav3D-unique surface gives you better options:

- `LCM_BTDecorator_IsInCover` - composite SweepPenalty + LOS predicate
- `LCM_BTService_ThreatFieldSampler` - writes `CurrentThreatLevel` + `bInThreatZone` into BB every 0.2s; then a stock blackboard decorator can gate on `bInThreatZone`

CHAPTER 17

BT PerceptionRelay - bind `OnTargetPerceptionUpdated` directly

TL;DR: Unreal Engine's `UAIPerceptionComponent` already emits perception updates as a delegate. You don't need a custom BT service - just bind the delegate in your AI controller and write to a blackboard key.

17.1 Pattern (in your AIController subclass)

```
// In your AIController.h
UPROPERTY()
TObjectPtr<UAIPerceptionComponent> PerceptionComp;

UFUNCTION()
void OnPerceptionUpdate(AActor* Actor, FAIStimulus Stimulus);

// In your AIController.cpp BeginPlay
PerceptionComp = FindComponentByClass<UAIPerceptionComponent>();
if (PerceptionComp)
{
    PerceptionComp->OnTargetPerceptionUpdated.AddDynamic(
        this, &AYourAIController::OnPerceptionUpdate);
}

void AYourAIController::OnPerceptionUpdate(AActor* Actor, FAIStimulus Stimulus)
{
    if (UBlackboardComponent* BB = GetBlackboardComponent())
    {
        if (Stimulus.WasSuccessfullySensed())
        {
            BB->SetValueAsObject("ClosestThreat", Actor);
            BB->SetValueAsBool  ("bHasAnyTarget", true);
        }
        else
        {
            // Lost sight - leave ClosestThreat for memory pattern (see below)
            BB->SetValueAsBool("bHasAnyTarget", false);
        }
    }
}
```

That's the canonical UE5 way. Three lines in your AIController + a Blackboard decorator in the BT.

17.2 Why we didn't ship `LCM_BTService_PerceptionRelay`

A custom service that ticks each frame and re-queries `GetCurrentlyPerceivedActors` is less efficient than binding the **event-driven** `OnTargetPerceptionUpdated` delegate. The stock pattern is also more idiomatic - every UE5 AI tutorial uses this pattern.

The LCM Nav3D StateTree variant `FLcmSTEval_PerceptionRelay` is kept because StateTree has no per-state perception delegate (its evaluators are tick-based by design). On the BT side, delegate binding is the standard.

17.3 Memory pattern (last-known-location with age)

UE5's perception component already tracks last-known location + age per perceived actor via `FActorPerceptionInfo::GetLastStimulusLocation(float* OutAge)`. To use it:

```
if (const FActorPerceptionInfo* Info = PerceptionComp->GetActorInfo(*Actor))
{
    float Age = 0.f;
    FVector LastKnown = Info->GetLastStimulusLocation(&Age);
    if (Age < 8.f) // 8 sec memory window
    {
        BB->SetValueAsVector("LastKnownThreatLoc", LastKnown);
    }
}
```

See `BT_MemoryDecay_StockPattern.md` for the full pattern. **Don't** roll your own decay timer - perception's age tracking already does it for you.

17.4 Related LCM Nav3D-unique services

For perception-aware predicates that need **LCM Nav3D LOS** (volumetric, not raycast):

- `LCM_BTDecorator_ThreatVisibleViaPerception` - composite: perception sees + LCM Nav3D Lazy Theta* LOS is clear
- `LCM_BTService_SquadSnapshot` - aggregates perceived allies into squad centroid / cohesion radius

CHAPTER 18

BT MemoryDecay - use `FActorPerceptionInfo::GetLastStimulusLocation(&OutAge)`

TL;DR: Unreal Engine's `UAIPerceptionComponent` already tracks per-actor last-known stimulus location with age. No need to roll your own decay timer.

18.1 Pattern (in your AIController or a small custom service)

```
UAIPerceptionComponent* Perception = GetAIPerceptionComponent();
if (!Perception) return;

// Pull the last-known info for a specific actor (e.g. the threat you're tracking).
if (const FActorPerceptionInfo* Info = Perception->GetActorInfo(*ThreatActor))
{
    float Age = 0.0f;
    FVector LastKnown = Info->GetLastStimulusLocation(&Age);

    // Apply your project's memory window.
    if (Age < MemoryWindowSec)
    {
        Blackboard->SetValueAsVector("LastKnownThreatLoc", LastKnown);
        Blackboard->SetValueAsFloat ("MemoryAge",          Age);
        Blackboard->SetValueAsBool  ("bHasMemory",         true);
    }
    else
    {
        // Memory expired.
        Blackboard->SetValueAsBool("bHasMemory", false);
    }
}
```

That's it. Perception's age tracking handles refresh-on-stimulus + decay automatically.

18.2 How to integrate as a BT service

Wrap the snippet above in a `UBTService_BlueprintBase` subclass (Blueprint-only - no C++ needed):

1. Create a BP service. Override `Receive Tick`.
2. Inputs: `ThreatActor` (Object BB key) + `MemoryWindowSec` (float).
3. Inputs to write: `LastKnownThreatLoc` (Vector BB key) + `MemoryAge` (Float BB key) + `bHasMemory` (Bool BB key).
4. Tick body: get perception component → `GetActorInfo` → `GetLastStimulusLocation(&Age)` → branch on Age vs window → write BB keys.

Five nodes total. No C++ required.

18.3 Why we didn't ship `LCM_BTService_MemoryDecay`

The stock `FActorPerceptionInfo::GetLastStimulusLocation(&OutAge)` provides everything our planned MemoryDecay service would - location + age tracked **per sense, per actor**, automatically refreshed on stimulus and aged each frame.

Rolling our own decay timer would:

- Duplicate the perception component's bookkeeping
- Miss the per-sense granularity (perception tracks separately for Sight / Hearing / Damage / etc.)
- Force buyers to maintain two parallel "last seen" caches

The LCM Nav3D StateTree variant `FLcmSTEval_Memory`. Same conclusion on both sides.

18.4 Related LCM Nav3D-unique services

If you want **squad-level memory** (where multiple allies' perceived-actor lists are aggregated), use `LCM_BTService_SquadSnapshot` - that's a genuinely LCM Nav3D-additive surface.

Part IV

Reference

CHAPTER 19

LCM Nav3D - Editor Tools

Everything here lives behind the **Nav3D** button in the Level Editor toolbar. The panels are also under **Window** ▸ **Level Editor**.

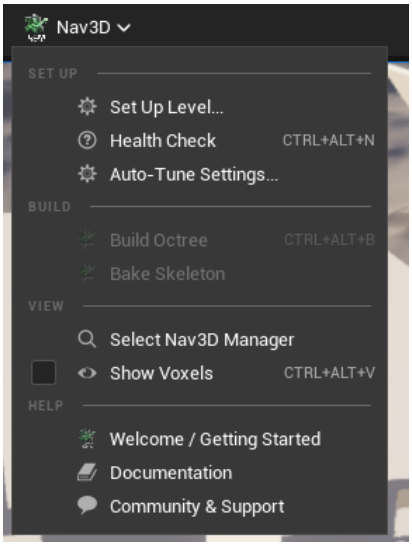


Figure 1 - The Nav3D toolbar menu. Four groups, in the order you need them. Greyed entries are not broken: **Build Octree** is finite-only and **Bake Skeleton** appears only when something will actually read a bake, so the menu never offers you an action that would be an expensive no-op.

19.1 Quick start

1. **Nav3D** ▸ **Set Up Level...** - creates and configures the manager, picks finite or streamed navigation to match your map, attaches navigation invokers, and builds.
2. **Nav3D** ▸ **Health Check** - dock it and leave it open while you work.
3. Add the **LCM Nav3D FlyTo** task to a Behavior Tree or StateTree, or drop the Mass crowd traits on a spawner.

Nav3D actors are also in the **Place Actors** panel under **LCM Nav3D**.

19.1.1 Keyboard shortcuts

Shortcut	Action
`Ctrl+Alt+N`	Health Check
`Ctrl+Alt+B`	Build Octree
`Ctrl+Alt+V`	Show Voxels

Rebind or clear them under **Editor Preferences** ▸ **Keyboard Shortcuts**.

19.2 The Welcome screen

Opens once per plugin version, and any time from **Nav3D ▸ Welcome / Getting Started**.

Figure 2 - Welcome screen. The part worth reading is **REQUIRED PLUGINS**: it checks the dependencies declared in `LCMNav3D.uplugin` against what your project actually has enabled, and offers an **Enable** button on any that are off. If one is missing, the plugin's modules will fail to load - this panel is where you find that out, rather than from a startup error.

> Links under **HELP & COMMUNITY** are read from the `.uplugin` descriptor, so the UI and the store listing cannot drift apart. A greyed link means that field is empty in the descriptor, not that the link is broken.

19.3 The Manager panel

Select the **LCM Nav3D Manager** in your level. Above the numbered settings sections you get a header card with live state pills (world model, residency, voxelizer), the context-appropriate build actions, and two readouts worth knowing about.

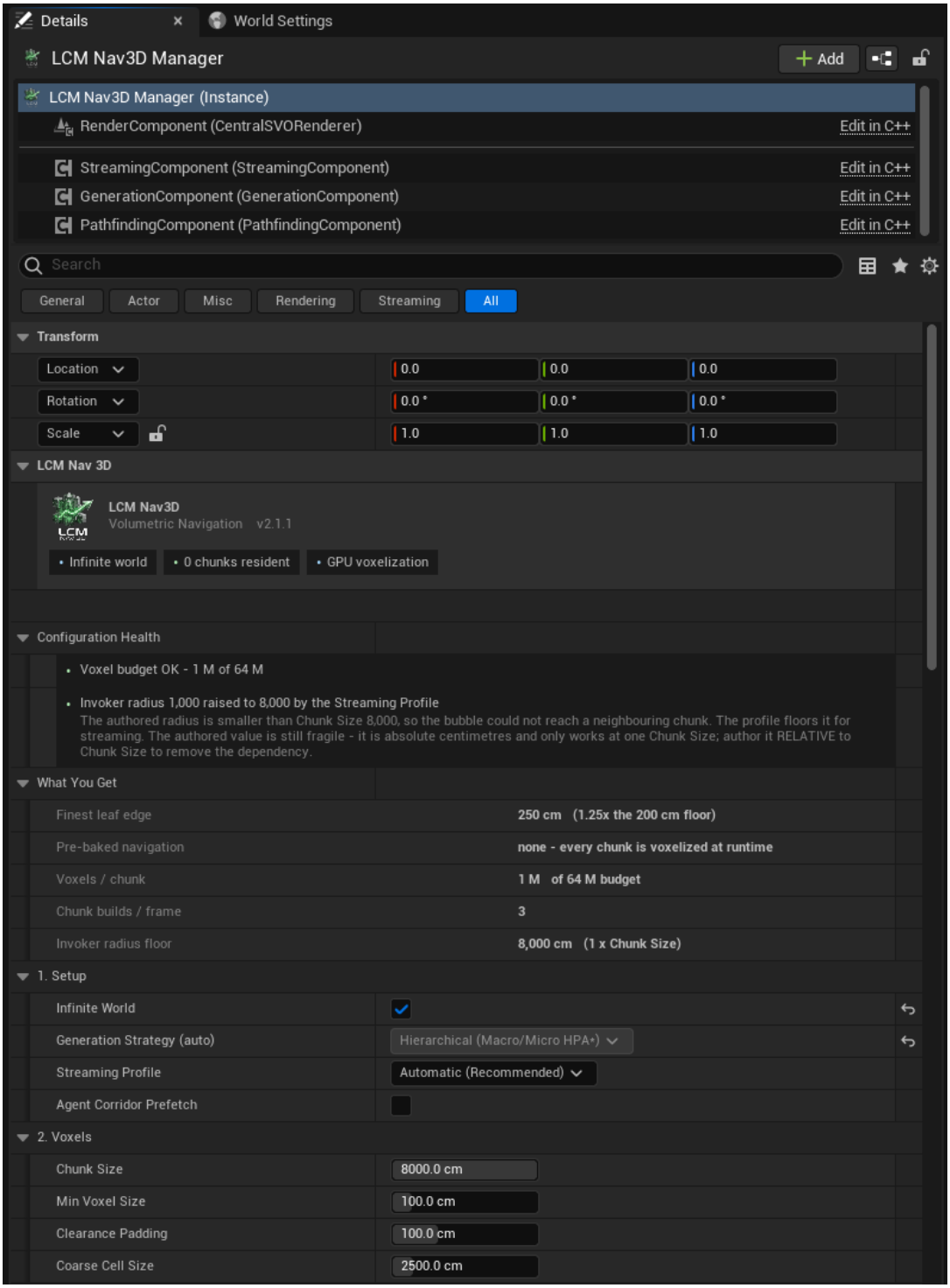


Figure 3 - Manager Details, top half. Read it downwards:

- **The pills** - world model, chunks resident, voxelizer. Three facts at a glance.
- **Configuration Health** - expands only while something is wrong. In this shot it is reporting that the authored invoker radius of 1,000 cm was *raised to 8,000* by the Streaming Profile, because a radius below one Chunk Size leaves an agent with no neighbouring chunk and therefore no route at all.
- **What You Get** - the derived numbers you would otherwise have to know a rule to predict. **Finest leaf edge 250 cm** with `Min Voxel Size` at 100 is not a bug: the octree halves a chunk until the child

drops below your voxel size, so the leaf is `ChunkSize / 2^k` and lands between 2× and 4× the value you typed.

19.3.1 The build actions

Only the buttons that mean something in your world model are shown - a finite world has no chunks to skeletonize, and a streamed world has no single SVO to save.

Button	World	What it does
Build SVO (Editor Preview)	Finite	Voxelizes now, into memory. EQS queries and the debug draw resolve without entering Play. **Thrown away when you close the editor** - it is a preview, not a build product
Bake to Asset	Finite	Writes the SVO to a `ULcmSVODataAsset` and attaches a streaming proxy that loads it. The shipped game stops voxelizing the world at every launch
Bake Skeleton	Infinite, and only when `Use Baked Skeleton` is on	Whole-map coarse connectivity, so long-range routing is complete from frame 0

Bake to Asset uses the manager's own `World Extent`, `Min Voxel Size`, `Clearance Padding` and collision filters, so the bake cannot describe a different world than the runtime expects. It records what it was baked *against*, which is what lets the health section tell you later that the bake has gone out of date. On success it also turns **Auto Build On Play** off: leaving both on is not belt-and-braces, because the proxy and the manager each write the navigation data during `BeginPlay` with no ordering guarantee between actors - which one wins is undefined, and when the rebuild wins you pay the full load-time stall the bake exists to remove.

Re-bake after moving level geometry or changing the voxel settings.

> **Save the level after baking.** The asset is written to disk immediately, but the streaming proxy that *loads* it is a level actor. Close without saving and you are left with a bake nothing reads - and `Auto Build On Play` cleared on an actor that was discarded, so the level has no navigation at all.

> **The `LCM Nav3D Bake Volume` actor is the other way to bake**, and it is for a different job - see Bake Volumes below. For a single finite world, the manager button is simpler and cannot disagree with the runtime settings. Both write the same asset format through the same code path.

19.3.2 Configuration Health

Setting *combinations* that no per-property clamp can express, checked live and shown where the settings are. Rows come in three tiers: ⚠ a real hazard, **i** a property worth knowing, ✓ an all-clear.

⚠ Warning	Why it matters
Voxel budget	Exceeding it does **not** fail - generation silently coarsens `Min Voxel Size` until it fits, so you get less precision than you configured
Clearance Padding vs Min Voxel Size	Padding inflates every solid surface, so padding ≥ 2× voxel size seals narrow openings. The right value is your agent **radius**, which only you know
Invoker radius vs Chunk Size	A radius under one `Chunk Size` leaves an agent with no neighbouring chunks, so no route can be planned and nothing further streams

⚠ Warning	Why it matters
Residency demand	An invoker bubble is a cube: chunks scale as `(2R / ChunkSize)³`, so halving `Chunk Size` costs 8× the chunks
Baked asset out of date	A bake is a frozen snapshot and nothing at runtime can tell it stopped matching the level. It still loads - you just get agents avoiding open space or routing through geometry, with nothing logged
Auto Build On Play overrides the bake	Both write the navigation data during `BeginPlay` with no ordering guarantee, so which one the level uses is undefined

i Information	What it tells you
Finest leaf vs the floor	The finest leaf is `ChunkSize / 2^k`, landing anywhere in [2×, 4×] `Min Voxel Size` **non-monotonically**. Not a defect - see below

> **Why the leaf ratio is information, not a warning.** It depends only on `Chunk Size` (it is identical at every `Min Voxel Size`), so it really asks "is Chunk Size a power-of-two multiple of the voxel floor" - and the Auto-Tuner's own candidate grid fails that test on most of its entries. Whether a coarse leaf actually *hurts* - a sealed opening, an unroutable pair - cannot be decided from the settings, and is exactly what the tuner measures. So the panel states the fact and lets the measurement carry the verdict.

> **What the staleness check can and cannot see.** It compares the voxel settings the bake ran with, and an order-independent hash of every collidable actor inside the baked bounds - identity, transform, scale and local bounds. So it catches geometry **moved, added, removed or rescaled**, which is nearly always what "the level changed" means. It does **not** walk source triangles, so a mesh edited in place - same actor, same transform, new geometry - hashes identically. Re-bake after editing a mesh. Assets baked before this signature existed report as unverifiable rather than stale.

When a ⚠ row is present the section offers one action: **Tune Settings...**, which opens the Auto-Tuner. There are deliberately no per-warning "apply this value" buttons - such a value would be *derived*, and this system's search space is non-monotonic, so only measurement can answer it.

These are the same checks that have always been written to the Output Log - the log is now a consumer of the same rules, so the two can never disagree.

19.3.3 What You Get

The values your settings actually resolve to, rather than the ones you typed:

Row	Note
`Navigable volume`	**Finite only.** The real world-space bounds - see the note below
`Finest leaf edge`	The real smallest voxel, with its ratio to the achievable floor (`2 × Min Voxel Size`)
`Voxels / chunk` (or `/ world`)	Against the hard budget
`Chunk builds / frame`	Derived from the Streaming Profile
`Invoker radius floor`	What the profile raises an invoker's radius to, if anything

> **The finite volume is centred on the world origin, not on the manager actor.** `World Extent` is a half-extent, so the volume spans `[-WorldExtent, +WorldExtent]` on each axis around `(0,0,0)`. The

> manager is a logical singleton with no root component, so **moving it does not move the navigable volume**. Build your playable space around the origin, or use an infinite world.

19.3.4 Why the panel is short

The manager has no geometry, no collision and no transform, so the inherited Actor sections that cannot apply - Rendering, Collision, Physics, LOD, Cooking, Input, HLOD, Mobile, Ray Tracing, Texture Streaming, Virtual Texture - are hidden from this panel. They still exist on the class; they are just noise here.

Replication, **Actor** (tags) and **World Partition** are deliberately kept: the manager hosts multicast RPCs, tags are a legitimate way to find it from gameplay, and `Is Spatially Loaded` genuinely matters for a singleton that must not stream out.

19.4 Bake Volumes

Drop an **LCM Nav3D Bake Volume** from the Place Actors panel and resize the brush to cover the area you want baked. Its panel uses the same language as the manager's - header pills, one action, Configuration Health, What You Get.

19.4.1 I want two separate areas navigable. Which setup?

Grid mode + Infinite World. Nothing else does it. This is the single most common wrong turn, so it is worth being blunt about why:

A **finite** world has exactly **one** navigation volume - one slot, in memory. A single-volume Bake Volume writes it. The manager's **Bake to Asset** writes it. Both write it at the same chunk coordinate, so whichever ran **last** wins and the other area has no navigation at all. **Overlapping the two does not join them** - there is nothing in a finite world that could hold both. Baking area A, then baking area B, and finding only B navigable is the system working exactly as a finite world is defined, not a bug in the bake.

What you want	Setup
One area	Either producer - a single-volume Bake Volume **or** the manager's Bake to Asset. **Not both**
One bigger area	Enlarge the volume, or raise the manager's `World Extent`
Two or more separate areas	`Snap To Global Grid` **on** each volume + `Infinite World` **on** the manager

The manager's Bake to Asset now **refuses** to run while a single-volume Bake Volume is in the level, rather than silently replacing its output, and both Details panels flag the pair.

19.4.2 One volume or many?

This is the `Snap To Global Grid` switch, and the answer differs because the two settings feed **different runtime containers**:

`Snap To Global Grid`	Produces	Manager must be	How many volumes
Off	One chunk, sized to this volume	**Finite** world	**Exactly one**
On	One chunk per grid cell	**Infinite** world	As many as you like

Off feeds the manager's single navigation slot, so a second single-volume bake has nowhere to go: both write the same asset name at the same chunk coordinate, and only the last one baked survives. To cover more ground with one volume, enlarge it - or raise the manager's `World Extent`.

On feeds a coordinate-keyed container that holds any number of chunks. This is the multi-region workflow, and it needs `Infinite World` on the manager. With a *finite* manager every proxy is routed into the one slot instead, so each chunk silently overwrites the last. The header pill turns amber when the two disagree, and Configuration Health says which way to resolve it.

> "Infinite World" names how navigation is *stored and streamed*, not whether the data is generated at runtime. A fully baked, grid-aligned world is still an infinite-mode world - the chunks just arrive from disk instead of the voxelizer.

19.4.3 Do several connected volumes route across their seams?

Yes. Chunks are keyed by grid coordinate, so which volume produced a chunk does not matter; adjacent chunks connect the same way whether they came from one volume or four. What matters is that every volume uses the **same** `Chunk Size`, `Min Voxel Size` and `Clearance Padding` (the panel checks all three), and that they are grid-aligned, which `Snap To Global Grid` guarantees.

Seams are **scanned at bake time** and shipped with each chunk, so long-range routing across a chunk boundary is correct from the first frame.

> **Assets baked before 2026-08-03 are different, and the panel will tell you.** They carry no seam data, so each boundary loaded as *one full-face portal* - a portal asserting the whole shared face was passable without checking the voxels, corrected only once an agent physically entered the chunk. A walled seam therefore looked *open* to long-range routing. The failure was a route that exists on the macro graph but not in the world, which is why it survived casual testing. **Re-bake to fix it**; Configuration Health reports how many chunks are still in that state.

Very large grids are the one exception. Scanning seams needs the whole grid resident at once (a portal is a property of the boundary *between* two chunks), so past `r.LcmNav.Bake.MaxMacroChunks` (default 1024) the bake falls back to writing chunks one at a time with no seam data. That is logged, and the completion dialog says so. Raise the cap if you have the memory, or bake in smaller volumes.

19.4.4 What the volume panel checks

Every rule here is about **two actors disagreeing**. The bake volume carries its own copy of the voxel settings and nothing reconciles them at runtime, so a mismatch does not fail - it bakes, it loads, and navigation is computed against a world that is not there.

⚠ Warning	Why it matters
Bake mode vs manager world model	Decides whether the chunks land in the single slot or the keyed container - the table above
`Chunk Size` ≠ the manager's	Baked chunks are placed at `coordinate × the runtime Chunk Size`, so a mismatch puts every chunk in the wrong place *and* makes the streamer regenerate the map at load
`Min Voxel Size` / `Clearance Padding` ≠ the manager's	The bake describes a differently-shaped world, and the staleness check compares against the *manager's* values - so the bake reads as out of date the moment it is written
Two single-volume bake volumes	Same asset name, same coordinate: silent overwrite
Voxel budget per chunk	Exceeding it coarsens `Min Voxel Size` silently, which is what voxelizes narrow openings shut
More than 512 chunks	The bake is synchronous on the game thread with no cancel, and writes one asset each
Baked chunks without precomputed seams	Assets from before 2026-08-03, or past the macro-bake cap - see the seam section above. Re-bake
The manager also bakes this volume	Two producers, one finite

slot - see the top of this section | | Oblong brush in single-volume mode | An octree root is a **cube**, so the bake inflates to the largest axis on all three and pays voxels for the remainder |

19.4.5 Sizing the brush, and why the numbers don't match it

Three things surprise people here, so the panel states all three under **What You Get**:

- **`Brush Settings` size × `Transform` scale.** Both scale the volume, and the readout shows the product. It is one box, controlled from two places.
- **Single-volume mode bakes a cube.** An octree root cannot be oblong, so the bake uses the **largest** half-extent on every axis: a 100 × 100 × 10 m brush bakes a 100 m cube, and the voxel budget is charged for all of it. That is why the budget row can dwarf the box you drew. Keep the brush roughly cubic, or switch to grid mode.
- **Grid mode snaps outward.** The brush is sliced on the *global* chunk lattice, so the baked region grows to whole chunk boundaries - straddling one by a centimetre costs an entire extra row of chunks. The **Actually voxelized** row shows the real region and count.

`Chunk Size` is hidden unless `Snap To Global Grid` is on, because it is not read otherwise - resolution comes from `Min Voxel Size` in both modes. When it *is* shown, it is snapped to a whole multiple of `Min Voxel Size` as you type (chunk edges must land on voxel boundaries), so the panel reports the value actually in use, and flags it if the manager disagrees.

i Information	What it tells you
No obstacle channels set	Empty falls back to `WorldStatic` **only**; anything solid on another channel bakes as open air

There is deliberately **no fix button** here. Every warning is two settings disagreeing, and either side may be the one that is wrong - that is a choice only you can make.

19.4.6 After baking, the debug view shows the result

Baking from a volume used to draw nothing while the manager's own build drew immediately, which looked like the bake had failed. It had not - the debug renderer reads the manager's *in-memory* navigation, and a bake writes assets and spawns proxies that inject nothing until `BeginPlay`. The bake now loads its own output back into the editor manager, so both paths look the same. Tick `Draw Debug Volumes` on the manager to see it.

19.5 Setup Wizard

One page, because every control feeds the same live readout.

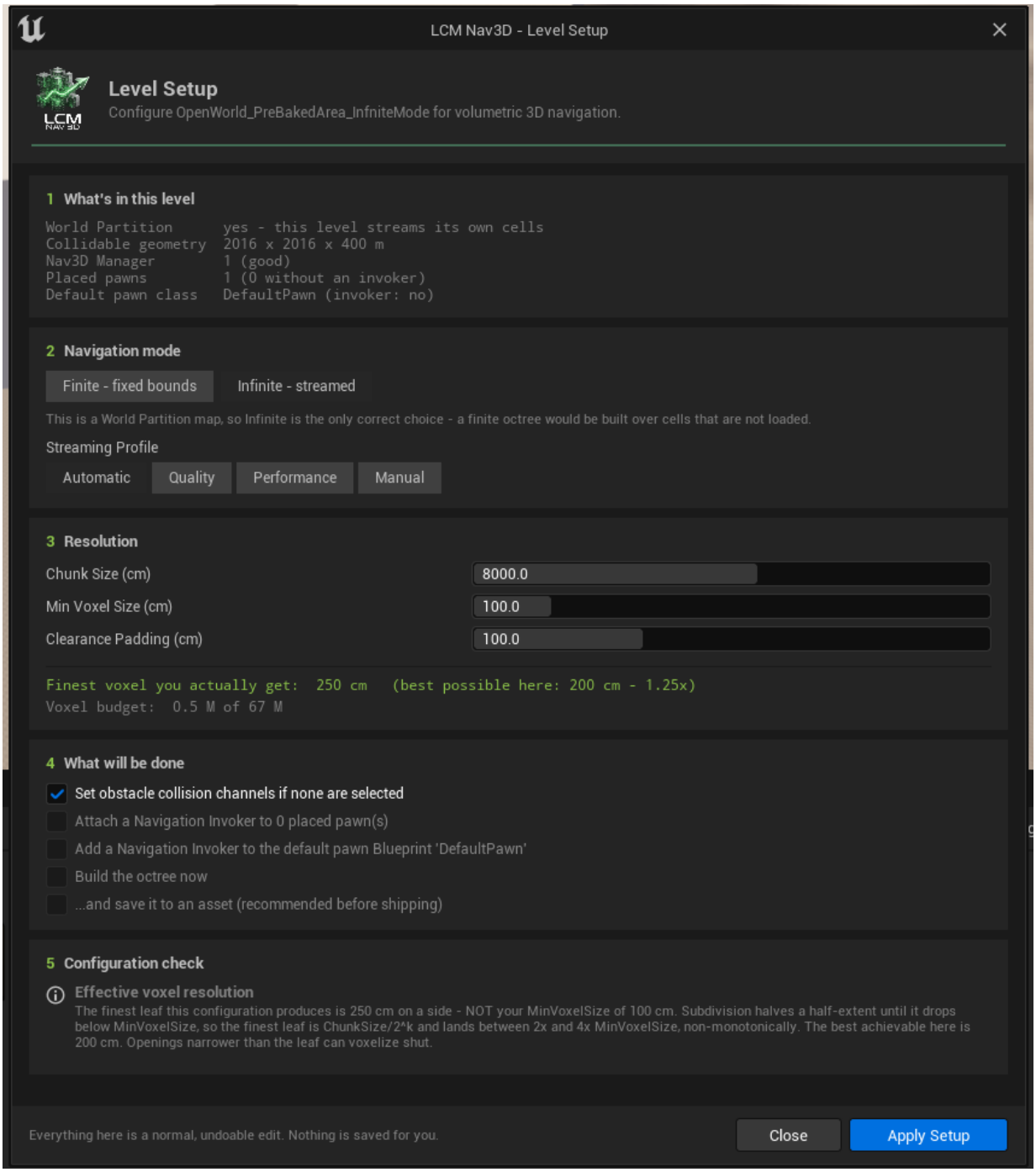


Figure 4 - Setup Wizard. Note what section 1 has already worked out on its own: this is a World Partition map, so section 2 tells you Infinite is the only correct choice and says *why* - a finite octree would be built over cells that are not loaded. Section 4 is an explicit checklist, so nothing happens that is not written on screen, and section 5 re-runs the configuration check against the settings you are *proposing* rather than the ones currently saved.

Section	What it does
What's in this level	World Partition, geometry size, manager count, pawns, the GameMode's default pawn
Navigation mode	Finite vs Infinite, plus the Streaming Profile
Resolution	Chunk Size / World Extent, Min Voxel Size, Clearance Padding
What will be done	An explicit checklist - nothing happens that isn't listed

Section	What it does
Configuration check	Live problems, updating as you change settings

Under **What will be done**, `Build the octree now` has a companion: **...and save it to an asset**. That is the same voxelization with its result *saved* rather than discarded - the wizard's route to the Bake to Asset flow. It is off by default, unlike the other actions: those are reversible edits to settings, while this one writes content to `/Game/LcmNavigation/BakedData` and adds an actor to your level. Save the level afterwards.

19.5.1 The resolution readout

```
Finest voxel you actually get: 250 cm (best possible here: 200 cm - 1.25x)
```

This is the most important line in the wizard. The finest leaf is `ChunkSize / 2^k`, so it lands between **2x and 4x** your Min Voxel Size and moves **non-monotonically** with Chunk Size. Measured at Min Voxel Size 100:

Chunk Size	Finest leaf
3000	375 cm
5000	312 cm
7000	219 cm
8000	250 cm

A smaller chunk can give you a **coarser** voxel. Openings narrower than the leaf voxelize shut, which is why "it works at one Chunk Size and not another" is a real effect and not a bug in your level.

Everything the wizard does is one undo step. Nothing is saved for you.

19.6 Health Check

A dockable panel that reports anything which would stop agents moving, with a **Fix** button on each row.

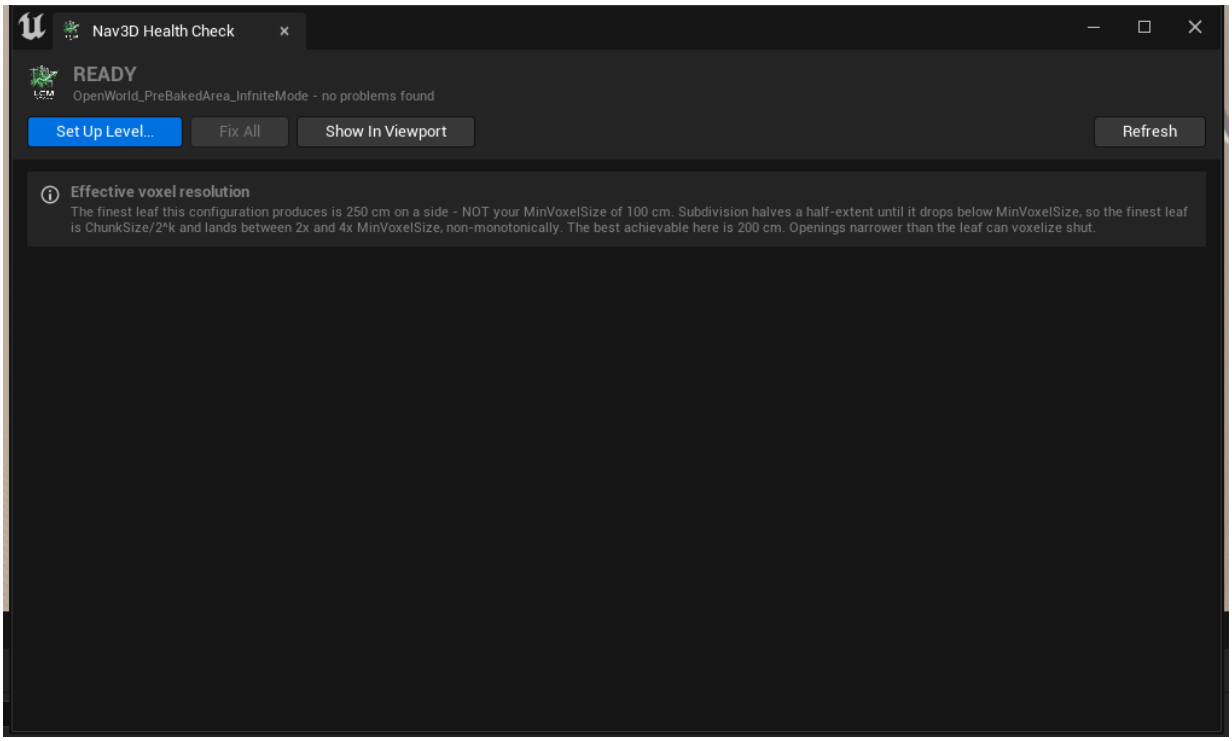


Figure 5 - Health Check, on a level that passes. The verdict is one word - **READY**, **WORKS, WITH WARNINGS** or **NOT READY** - coloured and sized so it reads from across the room. `Fix All` greys out when there is nothing to bulk-fix, which is the state you want to end up in and then stop thinking about. Dock this panel and leave it open; it re-inspects itself whenever the map, its actors or any Nav3D setting changes.

- **Fix All** applies every automatic repair, *except* ones that edit a Blueprint asset - those keep their own button, so changing your content is always a deliberate click.
- **Show In Viewport** draws the findings that have a *position*, for 30 seconds: the finite navigable volume in blue, your level's geometry bounds in red when they extend past it, and a marker over every pawn missing a Navigation Invoker. Read-only; it changes nothing.
- Refresh is automatic on map open, actor add/delete, and Nav3D setting changes.

> ★ **The volume box is worth looking at once.** `World Extent` is a *half* extent and the box is centred on the **world origin**, not on the Manager actor - the Manager has no root component, so dragging it 500 m away moves nothing. Everyone reads that in the tooltip; nobody believes it until they see the box sitting somewhere other than around their level.

Typical findings: no manager, more than one manager, a World Partition map set to Finite, no navigation invoker anywhere, over the voxel budget, Clearance Padding wide enough to seal openings.

> **Invoker radius below one Chunk Size** is the classic one. The agent has no neighbouring chunks, so no route can be planned, so nothing further streams - and it looks exactly like "the AI refuses to move". The Automatic, Quality and Performance streaming profiles floor it for you; Manual does not.

19.7 Auto-Tuner

Builds the octree at each candidate setting, runs **your** routes through the real planner, and ranks what it measured.

It runs as three steps, shown as a rail across the top:

1 Routes to test → 2 What to measure → 3 Measured results

- **Back is always available.** The loop is measure → look → adjust → measure again, so going back to fix a route is never punished.
- **Forward is gated**, and a disabled **Next** always states why in the footer: no routes yet, no Nav3D Manager in the level, or Play is running.
- A step is **ticked** in the rail once it is done. Step 3 only ticks when a configuration was actually *recommended* - a sweep where nothing completed every route is a finished run and an unfinished job.
- On step 2 the footer button *is* the run: **Run Measurement**, which advances to the results when it finishes. On step 3 it becomes **Measure Again**.

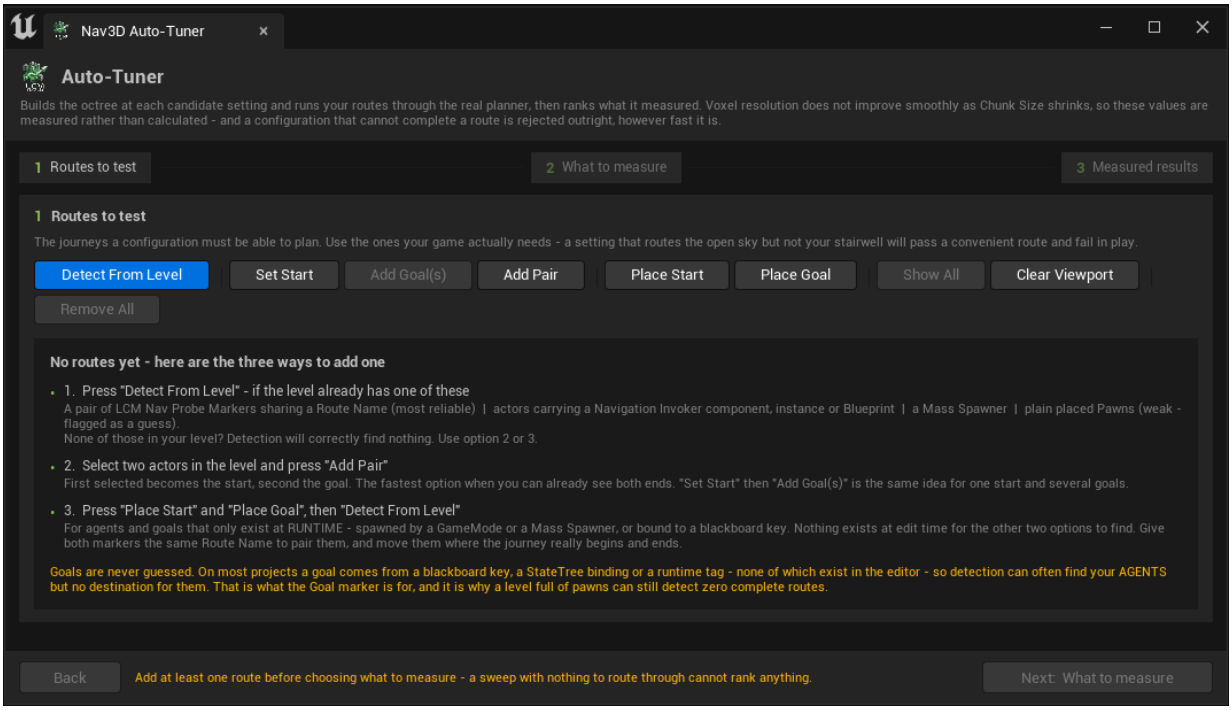


Figure 6 - Step 1, before you have added anything. The empty state is the screen every user sees first, so it does the teaching: three numbered ways in, each naming *what must already be in your level* for it to work. Note the footer - the **Next** button is disabled and the reason is printed beside it in amber rather than hidden in a tooltip.

19.7.1 Choosing routes

A route is a journey your agents actually make. **Every one must complete** for a configuration to be considered, so use the journeys your game depends on - a setting that routes the open sky but not your stairwell will pass a convenient route and fail in play.

Start with **Detect From Level**. The toolbar above the list is grouped by intent, left to right:

Group	Buttons	When
Find them for me	**Detect From Level**	Always try this first. Appends; never replaces hand-built routes, and skips duplicates.
Build from selection	**Set Start**, **Add Goal(s)**, **Add Pair**	You can see the endpoints in the level.
Place by hand	**Place Start**, **Place Goal**	Agents or goals only exist at runtime.
View	**Show All**, **Clear Viewport**	Check the endpoints before spending minutes on a sweep.
Discard	**Remove All**	-

19.7.2 What "Detect From Level" needs

Detection reads specific things out of the level. If none of them are there it correctly finds nothing - so this is the checklist for making it work, in descending order of confidence:

It looks for	You get
A pair of **LCM Nav Probe Markers** sharing a Route Name	The most reliable result - you stated the route explicitly
Actors carrying a **Navigation Invoker** component (instance *or* Blueprint)	Start points, from the actual Nav3D agent signal
An **AMassSpawner**	A start at the spawner's location
Plain placed **Pawns**	Starts, flagged as a *weak* guess

> ★ **Goals are never guessed, and this is the usual reason detection "does nothing".** On most projects a goal comes from a blackboard key, a StateTree binding or a runtime tag - none of which exist at edit time. So a level full of pawns can detect plenty of *agents* and still produce zero complete routes. That is exactly what the **Goal marker** is for.

When detection adds nothing, the panel says which of those cases you are in and what to do about it, rather than reporting "0 routes added".

Detection **appends** - it never replaces routes you built by hand, and it skips duplicates by endpoint.

19.7.3 Building routes by hand

- **Add Pair** - select exactly two actors; the first becomes the start, the second the goal. The fastest option when you can already see both ends.
- **Set Start / Add Goal(s)** - pin one start, then add many goals against it. While a start is pinned the panel shows a chip naming it, with an **Unpin** button; `Add Goal(s)` is disabled until one is pinned.
- **Probe markers** - place a **Start** and a **Goal** marker and give both the same **Route Name**, then press Detect. This is how you tune projects whose agents and goals only exist at runtime. An unpaired Goal marker pairs with every detected agent.

Each route carries its **own agent width** (diameter, cm), editable on its row - a gap a drone fits through may be unusable for a gunship, so one sweep can answer for a mixed fleet. `0` applies no size filtering.

Press **Show** on any route to highlight it in the viewport with the others drawn faintly for context.

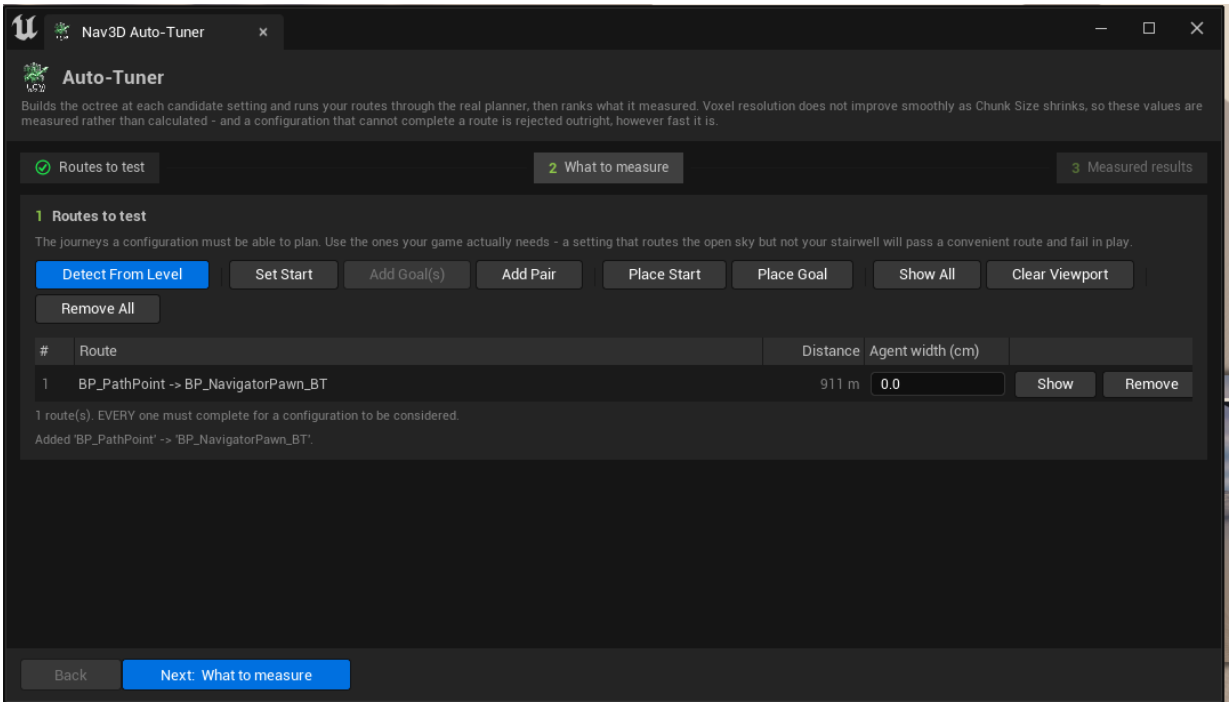


Figure 7 - Step 1, with a route added. `Add Pair` named it after the two actors it came from, in order, so a reversed route is obvious rather than something you debug later. Two things changed in the chrome the moment the route existed: step 1 in the rail gained a **tick**, and the footer's **Next** turned live.

19.7.4 Step 2 - What to measure

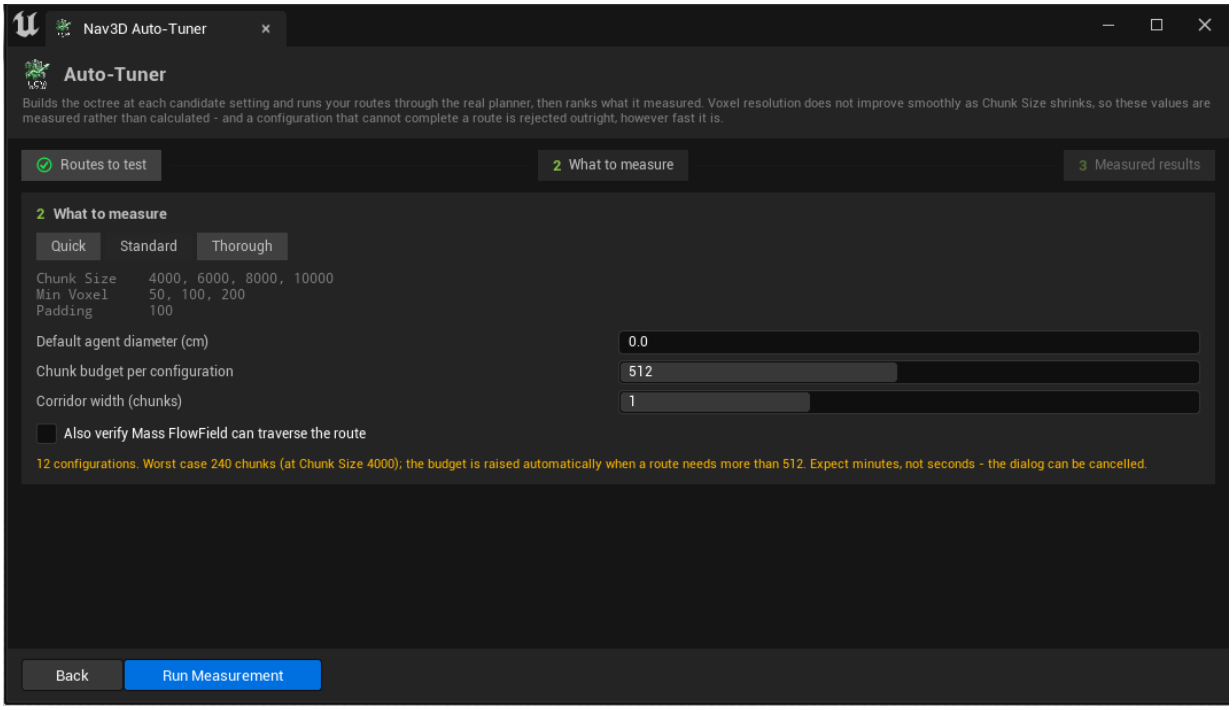


Figure 8 - Step 2. A preset never leaves you guessing what it means: the exact Chunk Size, Min Voxel and Padding values it will try are printed underneath it. The amber line is the honest cost estimate - configuration count, worst-case chunk count and *which* Chunk Size produces that worst case, because a smaller chunk needs more of them for the same route.

Leave **Also verify Mass FlowField can traverse the route** off unless you ship a FlowField crowd. Behaviour Tree, StateTree and Mass *PathFollowing* all dispatch through the same planner the tuner already queries,

so they are covered either way; FlowField is a different solver and costs an extra field solve per chunk per route.

19.7.5 How it ranks

1. **Completeness is a hard gate.** A configuration that cannot plan a route is rejected however fast it is. "Complete" means the path *reaches the goal*, not merely that the query returned success.
2. Among the survivors you get three picks - **Best Quality** (finest measured voxel), **Fastest** (cheapest to build and hold) and **Recommended** (the finest voxel within twice the cheapest one's memory).

The full matrix is always shown, so you can disagree on the merits. Click any column header to sort by it - the three picks keep their badges wherever they land, and sorting back to no column restores the ranked order. **Copy Table** puts every measured row on the clipboard as tab-separated text, which is a far better thing to attach to a support thread than a screenshot.

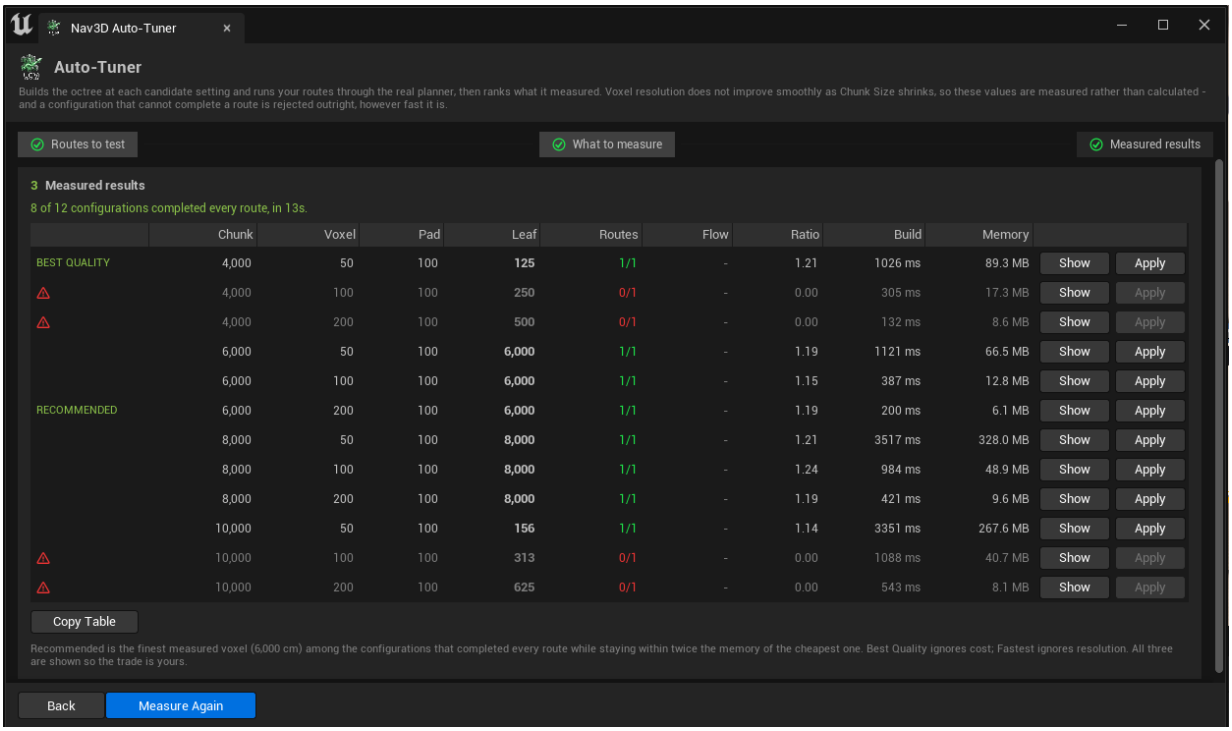


Figure 9 - Step 3, measured results. How to read a row:

Column	Read it as
Routes	The hard gate. Green `1/1` survived; red `0/1` is out no matter how good the rest of the row looks.
Leaf	The measured finest voxel - the number that decides what fits through a gap. Sort by it to rank quality.
Ratio	Path length ÷ straight-line distance. `1.00` is a straight line; nothing can be lower.
Build / Memory	What the configuration costs to produce and to hold. Sort by Memory to rank cost.
⚠ icon	This row could not route, or its build hit the chunk budget. Hover the row for the reason.

Show is enabled on failed rows too, where **Apply** is not. That asymmetry is deliberate: applying a broken configuration would be wrong, but *looking* at one is the whole point - the red rings it draws mark where each route gave up, and that position is usually the answer.

19.7.6 Seeing the routes

A route you cannot see is a route you cannot check, and the single mistake that invalidates a whole sweep - an endpoint sitting inside a building or under the terrain - measures as *"no configuration can route this level"*, which looks exactly like a real finding.

- **Show In Viewport** (routes card) draws every route for 30 seconds: **sphere = start**, **box = goal**, thin grey line between them. The line is the straight line, not a path - it makes no claim that a route exists.
- **Show** (on any measured row) draws the waypoints that configuration *actually* produced. Completed routes are green.
- A route that stopped short is marked with **red rings at the last waypoint**, and a thin line to the goal it never reached.

> ★ **The red rings are usually the whole answer.** A route that gives up against a doorway, vent or stairwell was not blocked by your level - it was blocked by **Clearance Padding**, which inflates every solid surface and seals openings narrower than the leaf voxel. Lower the padding and re-measure before concluding the level cannot be navigated.

`Show` is deliberately enabled on failed rows too, where `Apply` is not: applying a broken configuration would be wrong, but *looking* at why it broke is the point.

19.7.7 What it does and does not cover

Behavior Tree, StateTree, Mass **PathFollowing**	Covered - all use the same planner
Mass **FlowField**	Covered only with *"Also verify Mass FlowField..."* ticked - it is a different solver
Streaming (invoker radius, builds per frame)	**Not tuned** - that is the Streaming Profile's job
Dynamic obstacles	Voxelized where they currently sit; their motion is **not** simulated

Park moving obstacles in the positions that most constrain navigation before tuning, and set each route's **agent diameter** to whatever actually flies it.

> A sweep voxelizes the level once per configuration. Expect minutes, not seconds. The progress dialog can be cancelled, and your settings are restored either way - nothing changes until you press **Apply** on a row.

19.8 Project Settings

Project Settings ▸ Plugins ▸ LCM Nav3D. Project-wide only - anything that can differ between two levels lives on the Manager actor, where it is saved with the level.

19.8.1 Content Locations

Setting	Default
Baked Navigation Path	`/Game/LcmNavigation/BakedData`
Baked Skeleton Path	`/Game/LcmNavSkeletons`

Changing a path does **not** move an existing bake, but it does not orphan one either: the loaders probe the configured path first and the shipped default after, so an already-baked level keeps loading. Re-bake to migrate it properly.

19.8.2 New Level Defaults

What the **Setup Wizard proposes** for a level with no manager yet - Chunk Size, World Extent, Min Voxel Size, Clearance Padding, Streaming Profile and the obstacle collision channels. Set these once and "the way we set up a level here" stops being a convention someone has to remember.

They are *proposals*. Measured level geometry still overrides the world extent, an existing manager's authored values still win, and nothing here rewrites a level that is already configured.

> If your project puts level geometry on a **custom collision channel**, adding it here is the difference between a one-time setup step and a bug on every new level. An empty channel list voxelizes the world as open air, and that failure is silent - open air passes every clearance filter and every quality metric.

19.8.3 Console Variable Overrides

A map of `r.LcmNav.*` / `r.LcmGPU.*` names to values, applied at startup. This is the escape hatch for the ~345 console variables the page deliberately does *not* curate: pinning one here puts it in `DefaultGame.ini`, so it is version-controlled, survives a restart, and is visible to the whole team.

- Applied at **project-setting priority** - they beat a variable's built-in default and *lose* to the command line or anything typed into the console, so a local debug override is never stamped back over.
- A name that is not a registered variable is **warned about** in the log, not silently ignored.

> Most of these variables exist to A/B a research question and their correct shipping value is the default. Check first whether the behaviour you want is already a real property on the Manager or on the FlyTo task - it usually is, and a property is saved with the asset that needs it instead of applied to everything.

19.9 Keyboard shortcuts

Shortcut	Action
`Ctrl+Alt+N`	Health Check
`Ctrl+Alt+B`	Build Octree
`Ctrl+Alt+V`	Show Voxels

Rebind or clear them under **Editor Preferences ▸ Keyboard Shortcuts**.

19.10 Where things are configured

Setting	Where
World type, voxel sizes, collision channels	Nav3D Manager, Details panel
Streaming aggressiveness	Manager ▸ Streaming Profile
Per-agent size, smoothing, solver	The FlyTo task / Mass trait
Output paths, new-level defaults, pinned CVars	Project Settings ▸ Plugins ▸ LCM Nav3D
Everything else	`r.LcmNav.*` console variables

CHAPTER 20

Settings Reference: `LcmNavigationManagerSVO`

Every field on the navigation manager, in the order the Details panel presents them. Defaults and limits below are the shipping values.

If you only read one thing: the four settings that decide whether the plugin works well in your level are **`Infinite World`**, **`World Extent / Chunk Size`**, **`Min Voxel Size`**, and **`Clearance Padding`**. Everything else is refinement.

Concepts behind these are explained in Tutorial 02: Core Concepts.

20.1 The panel at a glance

The manager's Details panel is one **LCM Nav3D** group floated above the inherited Actor sections, then seven numbered sections in the order below.

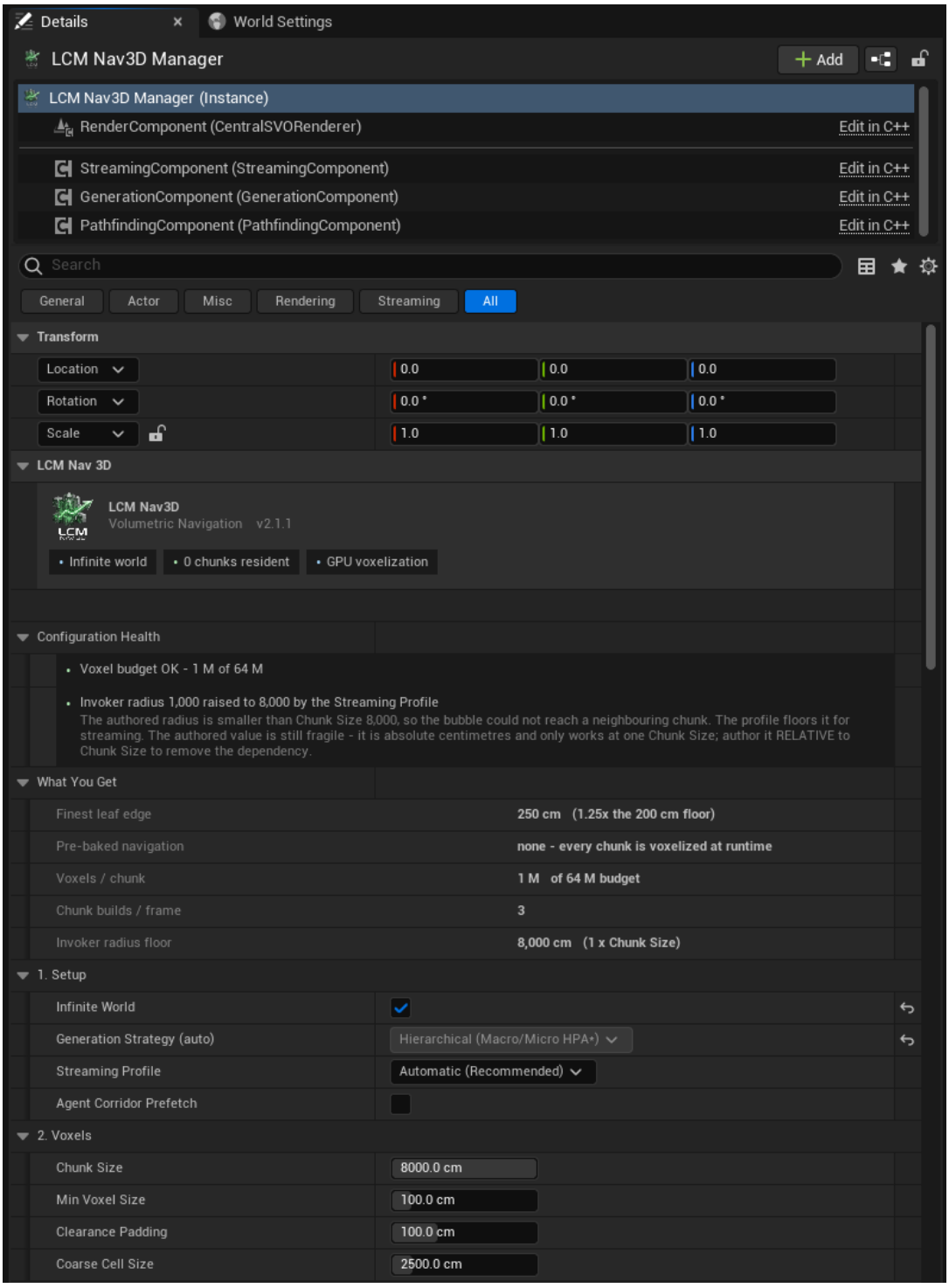


Figure 1 - Sections 1-2, plus the three blocks above them. *Configuration Health* and *What You Get* are not settings; they are what the settings below *resolve to*. They are worth reading before you change anything - the health block in this shot is reporting a raised invoker radius, and the *What You Get* block is reporting a 250 cm finest leaf from a 100 cm `Min Voxel Size`, which is the single most misread number in the plugin.

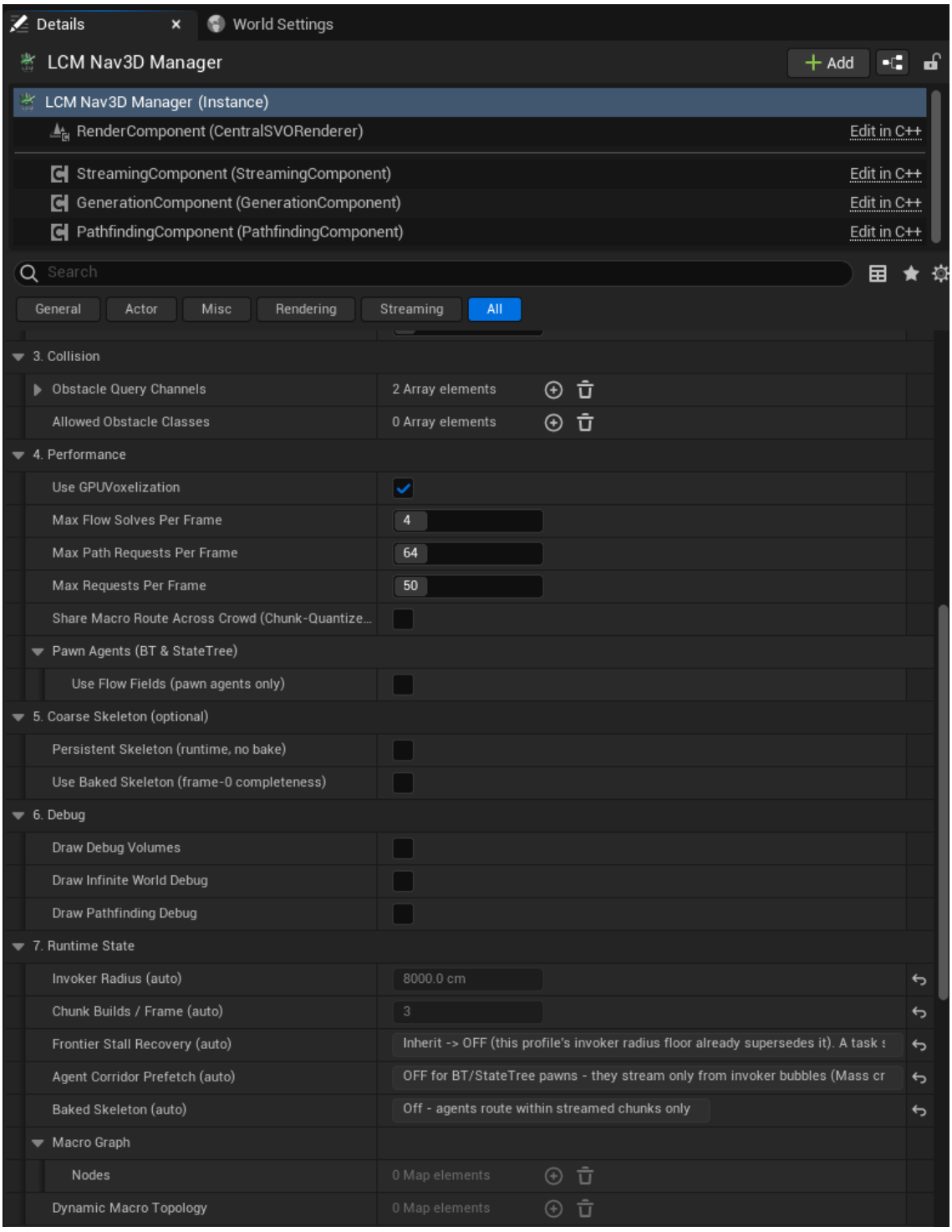


Figure 2 - Sections 3-7. Two things to notice:

- **Section 7 is read-only.** Every `(auto)` row is a value the Streaming Profile *derived*, shown so the profile is auditable rather than magic. `Frontier Stall Recovery (auto)` is the only place a BehaviorTree task's `Inherit` can be resolved for you, because a BT asset is not owned by a level and cannot know which manager it will run under.
- **Sections hide what does not apply.** A finite world shows no Coarse Skeleton options, and `Max Generations Per Frame` disappears unless you are finite or on the Manual profile - the plugin does not show you a property it is going to ignore.

20.2 1. Setup

20.2.1 `Infinite World`: default off

The biggest decision. Off = one fixed navigation volume built up front. On = the world is divided into chunks generated around agents as they move.

Leave it off unless you have a genuinely unbounded or streaming world. Finite is simpler, faster and more predictable.

Ticking this hides `World Extent` and reveals `Chunk Size` and the skeleton options.

20.2.2 `Auto Build On Play`: default on ***(finite only)***

Builds the navigation volume at `BeginPlay` - voxelizing the whole world, on every launch, on every player's machine. Turn it off if you want to trigger `BuildSVO` yourself from Blueprint at a specific moment, or if you have baked the SVO to an asset.

Turn it off when you bake. The **Bake to Asset** button does this for you, and it is not a tidiness measure: the streaming proxy and the manager both write the navigation data during `BeginPlay`, and actor order is not guaranteed, so with both enabled which one the level ends up using is undefined. The Configuration Health section flags the combination.

20.2.3 `Generation Strategy`: read-only ***(infinite only)***

Shows which streaming strategy was selected automatically.

20.3 2. Voxels

20.3.1 `World Extent`: default 10000 cm ***(finite only)***

Half-width of the navigation volume, measured outward from **the world origin**. The default gives a 200 m cube spanning `-10000 ... +10000` on each axis.

Range: 500-400000. Recommended UI range (slider): 2000-20000.

> **Anything outside this box has no navigation data.** Paths to targets outside it fail. Size it to cover your play space.

> **How big can it actually be? It depends on `Min Voxel Size`, not on a fixed distance.** The real limit is a voxel *count*: the world is `2 × World Extent` wide and holds `(2 × World Extent / Min Voxel Size)³` voxels against a 64 M ceiling - which is 400^3. So the largest legal half-extent is `200 × Min Voxel Size`:
>
> | `Min Voxel Size` | Largest `World Extent` | World size |
> |---|---|---|
> | 50 cm | 10 000 | 200 m cube |
> | 100 cm (default) | 20 000 | 400 m cube |
> | 200 cm | 40 000 | 800 m cube |
> | 500 cm | 100 000 | 2 km cube |
>
> The property clamp used to be a flat **20000**, which is the row for the *default* `Min Voxel Size` and wrong everywhere else - it blocked a perfectly legal 40 000 at 200 cm voxels, and allowed 8× over budget at 50 cm. A per-property clamp cannot express a rule that depends on another property, so the clamp is now just a sanity ceiling and **Configuration Health reports the real budget live** at whatever `Min Voxel Size` you are using. Exceeding it does not fail - the generator silently coarsens `Min Voxel Size` to fit, so you get less precision than you configured.
> Trading voxel precision for area is a real choice, not a workaround: a 2 km flying-combat arena at 500 cm voxels is a sensible configuration. Just make it deliberately.

> ⚠ **The volume does NOT follow the manager actor.** The manager is a logical singleton with no root component, so its position is ignored - moving it does not move the navigation volume. Build your playable space around the world origin, or switch to an infinite world. The manager's Details panel reports the resolved bounds under **What You Get** so you can see them directly.

20.3.2 `Chunk Size`: default 8000 cm *(infinite only)*

Edge length of one streaming chunk.

Range: 1000-800000. Recommended UI range (slider): 3000-8000. The clamp carries the same caveat as `World Extent` above - it is a sanity ceiling, not the real budget, which depends on `Min Voxel Size` and is reported live in Configuration Health. **Here the slider range is the number that matters:** it is the measured-reliable band, and going outside it degrades long-range routing regardless of the budget.

> ⚠ **Do not treat this as a free dial.** It affects memory, generation cost, *and* whether long routes can be found at all. **8000 is the tested default.** If you change it, re-test long-range routing specifically: a value that looks fine for short hops can break distant paths.

⚠ Chunk Size changes navigation *precision*, and not in the direction you expect

This is the single most surprising thing about the setting, so it is worth stating plainly.

Subdivision stops when a child's half-extent falls below `Min Voxel Size`, starting from `Chunk Size / 2`. The finest leaf edge you actually get is therefore `Chunk Size / 2`^{k} - it lands somewhere between **2× and 4× `Min Voxel Size`**, and it does **not** improve as Chunk Size shrinks. Measured on a real level at `Min Voxel Size = 100`:

Chunk Size	finest leaf edge	Chunk Size	finest leaf edge
3000	**375 cm** (3.75×)	7000	**219 cm** (2.19×)
5000	312 cm (3.12×)	8000	250 cm (2.50×)
6000	234 cm	9000	281 cm

So a smaller Chunk Size can give you a **coarser** octree. At `Chunk Size 3000` you get 375 cm leaves from a 100 cm setting - and **an opening that stays navigable at 250 cm can voxelize shut at 375 cm.** That is felt as *"the agent refuses to path"*, which reads like a bug rather than a setting.

What to do: leave it at 8000 unless you have a measured reason. If you must change it, use the **Auto-Tuner** (Editor Tools) - it *measures* the leaf edge from the built octree for each candidate rather than deriving it, and gates on whether a route exists at all before comparing quality or cost.

20.3.3 `Min Voxel Size`: default 100 cm

The finest resolution the octree will subdivide to. **The dominant quality/cost control.**

Range: 10-2000. Recommended UI range: 25-500.

- Smaller → finds tighter gaps; more memory; longer builds.
- Larger → cheaper; narrow openings disappear.

Rule of thumb: about **half the narrowest gap** agents must pass through.

> **It's a floor, not an exact size.** The octree halves down from the world/chunk size, so effective resolution snaps to powers of two. 100 → 90 may change nothing; 100 → 60 may double memory. Tune by testing.

20.3.4 `Clearance Padding`: default 100 cm

Inflates obstacles during voxelization so agents keep clear of surfaces.

Range: 0-1000. Recommended UI range: 0-300.

> ⚠ **Keep this well below `Min Voxel Size`.** Padding that approaches voxel size fills in doorways and corridors completely, and the pathfinder reports no route through openings you can clearly see. **This is the single most common cause of "it says there's no path but I can see the gap".**

20.3.5 `Coarse Cell Size`: default 2500 cm

Cell size of the coarse long-range routing layer. Larger = cheaper long-range planning, less precise. The default suits most levels.

Range: 100-20000.

20.4 3. Collision

20.4.1 `Obstacle Query Channels`

Which collision channels count as obstacles. Anything on these channels is voxelized as solid. Defaults to `WorldStatic` + `WorldDynamic`.

> **Empty means *nothing*, not *everything*.** Unlike `Allowed Obstacle Classes` below, clearing this list does not widen the query - it voxelizes an empty world, and you get navigation data that says every point is open air. That passes every metric and shows up only as agents flying through walls. **Bake to Asset** refuses to run on an empty list for exactly this reason.

> If a wall isn't blocking agents, check that its collision channel is in this list **and** that the mesh actually has collision enabled.

20.4.2 `Allowed Obstacle Classes`

Optional filter. Leave **empty** to consider all actors (normal). Populate it to restrict voxelization to specific actor classes (useful when you want to ignore scenery entirely).

20.5 4. Performance

20.5.1 `Use GPU Voxelization`: default on

Builds the octree on the GPU, which is substantially faster. Falls back to CPU automatically if no suitable GPU is present. Turn it off only to force deterministic, reproducible builds.

20.5.2 `Max Generations Per Frame`: default 1

Chunk builds allowed per frame. Higher fills the world faster but can cause hitches. *Range: 1+, UI up to 8.*

20.5.3 `Max Path Requests Per Frame`: default 64

Path queries dequeued per frame. Raise for many agents replanning at once; lower to protect frame time. *Range: 1-256.*

20.5.4 `Max Requests Per Frame`: default 50

General request throttle. *Range: 1-200.*

20.5.5 `Max Flow Solves Per Frame`: default 4

Flow-field solves per frame. *Range: 0-16.*

20.5.6 `Finite Obstacle Update Interval`: default 0.1 s ***(finite only)***

How often dynamic obstacle changes are folded into the octree. Lower = more reactive, more CPU. *Range: 0-2 s.* See Tutorial 05.

20.5.7 `Chunk Quantized Macro Cache`: default off

Caches long-range routes quantized to chunk boundaries. Helps when many agents share similar long-range routes (large crowds spawning together).

20.5.8 Pawn Agents (BT & StateTree) subcategory

Field	Default	Meaning
`Use Flow Fields (pawn agents only)`	**off**	Opt-in: pawn agents heading to a shared goal use one steering field instead of individual paths
`Flow Field Threshold`	`16`	Agent count at which a shared field is built *(range 2-512)*
`Flow Field Cluster Cell Size`	`1000` cm	Grouping granularity for shared goals *(range 100-10000)*

> Flow fields are for **swarms heading to the same place**. For a handful of agents with individual goals they are pure overhead, so leave it off.

20.6 5. Coarse Skeleton (optional)

Two independent, composable ways to keep **long-range** macro routing complete in a streamed world. **Both are opt-in and default off.** They are not part of the base configuration - turn one on only when an agent must route to somewhere it cannot currently see. Agents operating inside their streaming bubble need neither.

20.6.1 `Persistent Skeleton (runtime, no bake)`: default off ***(infinite only)***

Keeps a lightweight long-range connectivity graph in memory so distant routes can be planned before the detailed chunks exist. No bake required; works on any world, including procedural ones.

Turn it on when an agent **returns across ground it has already covered** and must not lose routability over it. The cost is memory that grows with how much ground has been covered - one compact node per visited-then-evicted chunk, bounded by `r.LcmNav.MaxSkeletonNodes`, not by the size of the resident set.

20.6.2 `Use Baked Skeleton (frame-0 completeness)`: default off ***(infinite only)***

Loads a pre-baked skeleton so long-range routing is complete from the very first frame, including into **never-visited** space.

Tick this on first. The **Bake Skeleton** button only appears in the banner once it is on - nothing else reads a bake, so the button stays hidden until you have asked for one. Then click it Rebake after changing the level or the voxel settings - a stale bake is refused at load, and the `Baked Skeleton (auto)` line under **7. Runtime State** will say so.

`Baked Skeleton (auto)` under **7. Runtime State** always reports which of the two, if either, is actually active.

20.7 6. Debug

20.7.1 `Draw Debug Volumes`: default off

Draws the octree. **The most useful diagnostic in the plugin**: if a gap shows no fine voxels, the pathfinder can't route through it either. Expensive; turn it off when you're done.

Also available from Blueprint via `SetDrawDebugVolumes` / `ToggleDrawDebugVolumes`.

20.7.2 `Draw Pathfinding Debug`: default off

Draws computed paths.

20.7.3 `Draw Infinite World Debug`: default off *(infinite only)*

Draws chunk boundaries and streaming state.

20.7.4 `Debug Redraw Coalesce Seconds`: default 0.25 s

How long debug redraws are batched. Raise it if debug drawing itself is hurting frame rate.

> **`Force Redraw SVO` was removed.** It was a checkbox that instantly unchecked itself, and it was destructive: on a level with no baked data it *deleted* the editor preview it was meant to refresh. Reloading a bake into the preview now happens automatically when you bake, and the manual action is the **Refresh Preview** button, which appears on the manager only when the level actually has baked navigation to reload.

20.8 7. Runtime State

Read-only diagnostics showing live counts of resident chunks, skeleton state and staleness. Useful when tuning an infinite world; nothing to set.

20.9 Build & Tools (Blueprint-callable)

Function	Purpose
`BuildSVO`	Build the navigation volume now
`BuildSVOInEditor`	Build in-editor **into memory**, so EQS previews work without playing. Not saved
`BakeFiniteSVOToAsset`	*(finite)* Save the SVO to an asset + streaming proxy, so the level stops voxelizing at launch. Turns `Auto Build On Play` off. Returns **false** and does nothing if a single-volume Bake Volume already owns this level's one finite volume
`BakeSkeletonInEditor`	*(infinite)* Pre-bake the long-range skeleton for `Use Baked Skeleton`
`LoadBakedSkeleton`	Load a previously baked skeleton
`FindPath`	Synchronous path query: start, end, options, out array of points
`FindCover`	Tactical cover query
`UpdateDynamicObstacle`	Manually stamp an obstacle by bounds
`SetMacroTopologyCell`	Manually mark a coarse cell blocked/free
`ForceRedrawSVO`, `SetDrawDebug Volumes`, `ToggleDrawDebugVolumes`, `DebugDrawFlowField`	Debug

20.10 Per-agent settings

These live on the **Fly To** BT task / **Fly To** StateTree task, not the manager. Both tasks expose the identical surface, in six numbered sections:

1. Goal

Field	Default	Meaning
Acceptance Radius	100 cm	Distance at which the goal counts as reached
Target Location	-	*(StateTree only)* world-space goal
Target Actor	*none*	*(StateTree only)* moving goal; overrides Target Location when chase is on

2. Movement

Field	Default	Shown when	Meaning
Use Flight Movement Component	off	always	Delegate the body to a ULcmFlightMovementComponent for 6-DOF drone / bird / fish dynamics
Flight Speed	600 cm/s	not delegating	Cruise speed
Max Acceleration	2000 cm/s^2	not delegating	Lower = heavier feel; also sets how hard the agent can brake into its arrival
Flight Style	*Physics (Drones/Ships)*	not delegating	vs *Linear (Biological/Magic)*
Deceleration Distance	800 cm	not delegating **and** Physics-weighted	Slow-down lead-in. LinearDirect has no inertia to bleed off, and the flight component brakes on `v = sqrt(2·a·d)` against the real goal distance instead
Banking Amount	4.0	not delegating	Roll into turns (cosmetic)
Max Banking Angle	45°	not delegating	Roll limit (cosmetic)

> **Whoever owns the body owns its caps.** Tick `Use Flight Movement Component` and this section collapses to that one checkbox: speed, acceleration, flight style and banking are set on the component's **Flight Config** instead, and the task reads them from there for its guidance, braking and chase-lead maths. Untick it and the task's own values apply. Exactly one place is ever live.

3. Pathfinding

Field	Default	Meaning
Query Options	-	Solver, heuristic, smoothing, agent size, wall-distance preference

4. Avoidance *(the reactive knobs hide when Avoidance Style is CBS)*

Field	Default	Meaning
Avoidance Style	*Reactive (Boids/Push)*	None / Reactive / Predictive (ORCA) / Context-Based

Field	Default	Meaning
Avoidance Time Horizon	2.0 s	ORCA look-ahead (Predictive only)
Collision Margin	1.2	Safety multiplier on the avoidance radius
Wall Check Distance	200 cm	Wall probe range
Wall Push Force	4000	Wall push strength - keep it strong
Path Centering Force	600	0 = free drift, high = train-on-tracks
Separation Radius	150 cm	Neighbour spacing
Separation Weight	2.0	Spacing strength

5. Moving Goal (Chase)

Field	Default	Meaning
Enable Goal Prediction	off	Intercept a moving target
Force Enable Chase	off	Activate chase without also setting r.LcmNav.Chase.Enable
Lead Seconds Cap	2.0 s	Prediction limit
Drift Replan Cm	400 cm	Target movement before replanning

6. Streaming *(infinite worlds only)*

Field	Default	Meaning
Frontier Stall Recovery	*Inherit*	Coast / climb out / arrive un-quantized when stalled at a streaming frontier. *Inherit* defers to the manager's Streaming Profile

> Frontier Stall Recovery is the **locomotion** half only. Pinning chunks *ahead* of the agent is a separate world-level setting - **Agent Corridor Prefetch** on the LCM Nav3D Manager. They used to be one checkbox named for the half it mostly wasn't.

Solver and smoothing choices are described in Tutorial 02 §6.

20.11 Recommended starting points

Scenario	Infinite	Extent / Chunk	Min Voxel Size	Clearance Padding
Arena or single building	off	10000	100	100
Tight interiors, small agents	off	10000	50	25
Large outdoor level	off	20000	200	100
Open / streaming world	**on**	8000	100	100

Start here, then confirm with Draw Debug Volumes that the gaps you care about actually have voxels in them.

CHAPTER 21

Troubleshooting

Problems in the order people actually hit them. **Start with the Quick Triage table**: it resolves most issues in under a minute.

21.1 Quick triage

Symptom	Most likely cause	Jump to
Agent doesn't move at all	No navigation manager in the level	§1
"No path" through a gap you can see	`Clearance Padding` sealed it	§3
Agent flies through walls	Geometry not voxelized	§4
Moving obstacles ignored	Missing Dynamic Obstacle component	§5
Can't find the demo maps	*Show Plugin Content* is off	§7
Frame rate drops	Voxel resolution too fine	§6
Long paths fail, short ones work	Infinite-world streaming	§3

21.2 The two diagnostics that answer almost everything

1. Turn on `Draw Debug Volumes` (navigation manager → *6. Debug*).

This draws the octree. It tells you what the pathfinder actually sees, which is frequently not what you assume:

- **No small voxels around a doorway** → the pathfinder can't route through it either.
- **No voxels at all in a region** → it's outside `World Extent`, or the chunk hasn't generated.
- **Solid voxels where there's visibly open air** → `Clearance Padding` is too large.

2. Check the Output Log (Window → Output Log) and filter for `Lcm`. The plugin logs the specific reason a query failed.

21.3 1. Nothing moves

Work in this order.

21.3.1 Is there a navigation manager?

Output Log shows **`LCMBTTask_FlyTo: No SVO Manager found in the level!`** → drag an **`LcmNavigationManagerSVO`** into the level. One per level, required.

21.3.2 Did the AI possess the pawn?

- Pawn → Class Defaults → **`Auto Possess AI`** = **Placed in World or Spawned**
- Pawn → Class Defaults → **`AI Controller Class`** is set
- The controller actually runs the tree (`Run Behavior Tree`, or a `StateTreeAIComponent` with a State Tree assigned)

21.3.3 Can the pawn move at all?

It needs a movement component that supports flight. **Floating Pawn Movement** is the simple choice. Check its **Max Speed** isn't 0.

21.3.4 Is the goal set?

Print the blackboard key (or StateTree parameter). A goal of `(0,0,0)` means it was never set, or was set after the tree started.

21.4 2. Agent moves, but badly

Symptom	Cause	Fix
Jitters, vibrates in place	Physics fighting navigation	Set mesh collision to *No Collision* to confirm, then re-add carefully
Stops short of the goal	`Acceptance Radius` too large	Lower it, but not below one voxel
Overshoots and circles	Speed too high for turn rate	Lower `Flight Speed` or raise `Deceleration Distance`
Robotic square turns	Grid-locked solver, no smoothing	Use *Lazy Theta** and *Curved Spline* smoothing
Sinks into the floor	Insufficient clearance below	Raise `Clearance Padding` slightly, or `Min Voxel Size` finer
Agents pile up on each other	No separation	Raise `Separation Radius` / `Separation Weight`, or use *Predictive (ORCA)* avoidance

21.5 3. "No path found"

21.5.1 First: is the goal reachable at all?

- **Inside the navigation volume?** Outside `World Extent` there is no data. Enlarge it, or move the goal in.
- **Inside geometry?** A goal (or start) buried in a wall has no free voxel. Move it into open air.
- **Genuinely enclosed?** If the destination is sealed off, failure is correct.

21.5.2 The classic: padding sealed the opening

This is the most common cause of a mystifying "no path".

If `Clearance Padding` is close to `Min Voxel Size`, obstacle inflation fills narrow openings completely. A 2 m doorway with 100 cm voxels and 100 cm padding can vanish entirely.

Fix: lower `Clearance Padding`, or lower `Min Voxel Size` so the opening survives the inflation. Confirm with `Draw Debug Volumes`: you should see free voxels *through* the opening.

21.5.3 Short paths work, long ones fail

Almost always an **infinite world** issue: the route crosses chunks that haven't generated yet.

- Turn on **`Persistent Skeleton`** under **5. Coarse Skeleton (optional)**. It is **off by default**
- it is what allows long-range planning before the detail exists, and it is the first thing to try when an agent loses routability over ground it has already crossed.
- For routing to be complete on the very first frame, including into space the agent has never visited, tick **`Use Baked Skeleton`** (also under **5. Coarse Skeleton (optional)**) and then click the **`Bake Skeleton`** button, which appears in the banner once that option is on.
- Tick **`Assist Infinite Streaming`** on the Fly To task for agents that must traverse the frontier.
- Leave **`Chunk Size`** at **8000** unless you have a specific reason and will re-test long routes.

21.5.4 Everything fails immediately

The volume never built. Check **`Auto Build On Play`** is on, or that you call `BuildSVO`. With `Draw Debug Volumes` on, you should see the octree at `BeginPlay`.

21.6 4. Agents ignore geometry

If agents fly through a wall, the wall isn't in the octree.

1. **Does the mesh have collision?** Voxelization uses collision, not visual geometry. *No Collision* = invisible to navigation.
2. **Is its channel in `Obstacle Query Channels`?** *(3. Collision)*
3. **Is `Allowed Obstacle Classes` filtering it out?** Empty = allow everything, which is what you normally want.
4. **Was it added after the build?** Static geometry is baked at build time. Something spawned later needs either a rebuild or a Dynamic Obstacle component.
5. **Is it too thin?** Geometry thinner than a voxel can be missed. Lower `Min Voxel Size`, or thicken the collision.

21.7 5. Dynamic obstacles do nothing

1. **Does the actor have an `LCM Nav3D Dynamic Obstacle` component?** Moving an actor without one changes nothing as far as navigation is concerned, and produces no error.
2. **Does its mesh have collision enabled?**
3. **Is `Update Distance Threshold` too high?** It won't restamp until the actor moves that far (default 50 cm).
4. **Reaction too slow?** Lower `Finite Obstacle Update Interval` (default 0.1 s).
5. **Using `Tracked Components`?** If the list is non-empty, only those components are tracked, so make sure the blocking one is in it.

See Tutorial 05.

21.8 6. Performance

21.8.1 Build is slow / hitches when streaming

- Raise **`Min Voxel Size`** (the biggest lever by far).
- Keep **`Use GPU Voxelization`** on.
- Lower **`Max Generations Per Frame`** to spread work (world fills in more slowly, but smoothly).

21.8.2 Frame rate drops with many agents

- Lower **`Max Path Requests Per Frame`** to cap per-frame planning.
- Use cheaper avoidance: *Reactive* rather than *Predictive*.
- For agents sharing a goal, enable **`Use Flow Fields`**.
- **Past a few hundred agents, move to Mass Entity.** Actor ticks, not pathfinding, are the ceiling.

21.8.3 Memory too high

- Raise **`Min Voxel Size`** (memory scales steeply with resolution).
- Shrink **`World Extent`** to just your play space.
- In infinite worlds, keep `Chunk Size` at the default.

21.8.4 Debug drawing is itself slow

Expected: it's a debug tool. Raise `Debug Redraw Coalesce Seconds`, or turn it off.

21.9 7. Editor and install

Problem	Fix
Can't find demo maps	Content Browser → Settings → **Show Plugin Content**
"Modules are missing or built with a different engine version"	Rebuild; if it fails, your project is Blueprint-only. See Tutorial 01
Plugin absent from the Plugins window	Wrong folder layout: must be `YourProject/Plugins/LCMNav3D/LCMNav3D.uplugin`
Fails to load, `GetLastError 126`	A Mass plugin dependency got disabled; re-enable `MassEntity`, `MassGameplay`, `MassAI`
Compile errors about StateTree or Mass types	Same cause: an engine plugin dependency is off
EQS preview shows nothing in-editor	Call **`Build SVO In Editor`** on the manager; EQS previews need navigation data without playing

21.10 8. Determinism and multiplayer

If you need identical results across runs (replays, lockstep, automated tests):

- Turn **`Use GPU Voxelization`** off.

Path solving already runs on the CPU and is deterministic by design. Voxelization is the part that can vary: GPU execution order isn't guaranteed run-to-run, so force the CPU voxelizer when you need a bit-identical rebuild.

21.11 Still stuck?

Gather these before asking for help, since they identify most problems immediately:

1. **`Draw Debug Volumes` screenshot** of the area that misbehaves.
2. **Output Log** filtered to `Lcm`.
3. **Manager settings**: `Infinite World`, `World Extent/Chunk Size`, `Min Voxel Size`, `Clearance Padding`.
4. **Which path** you're using: Behavior Tree, StateTree, or Mass.
5. **Does the equivalent demo map work?** If yes it's a setup issue in your level; if no it's an install issue.

Then bring them to the community Discord: **https://discord.gg/nXkpD6uaBG**

Open a **Technical Support** ticket in `#get-help` and you get a private channel with the author. First reply within 24 hours on weekdays. `#support-faq` there carries this same triage list, and `#known-issues` tracks confirmed defects with their workarounds - worth checking before you write.

CHAPTER 22

LCM Nav3D - Realistic Flight/Swim Locomotion Reference

Ultra-realistic 6-DOF locomotion for flying/swimming pawns (drones, birds, fish), layered on top of LCMNav3D navigation. The nav system provides **guidance** (where to go); this layer provides physically-plausible **dynamics** (how the body moves).

22.1 Concept - Guidance → Control → Dynamics

- **Nav (unchanged):** SVO path / flow field → pure-pursuit + CBS/ORCA avoidance → a *desired velocity*.
- **`ULcmFlightMovementComponent` (new):** consumes that desired velocity and runs a 6-DOF rigid-body flight/hydro model (thrust, lift/drag, gravity/buoyancy, attitude PID) per archetype, producing the pawn transform **and** an animation-drive struct.

22.2 Quick start

1. Add a **Lcm Flight Movement Component** to a pawn whose root is a collision primitive.
2. Set: `bEnabled = true`, `LocomotionModel = FlightDynamics`, `Archetype` (Multirotor / Bird / Fish / Generic6DOF), and tune `FlightConfig`.
3. Drive it one of two ways:
   - **Nav-driven (recommended):** on the pawn's **BT/StateTree FlyTo** task, tick **`Use Flight Movement Component`**. The task feeds guidance automatically.
   - **Manual:** call `RequestVelocity(WorldDesiredVelocity)` each tick from your own code/Blueprint.

Off by default: with `bEnabled=false` (or the task flag off) nothing changes - motion is the legacy kinematic FlyTo.

Already have a movement component on the pawn? Leave it there - nothing to remove. While the flight component is enabled it takes **sole movement authority**: any other movement component driving the same root (`FloatingPawnMovement`, `CharacterMovement`, `ULcmNavMovementComponent`) is automatically paused (velocity zeroed, deactivated, tick off) and restored exactly as found the moment flight stops driving (`bEnabled=false`, the component deactivated, or the pawn torn down). Without this, UE runs both integrators against one body and the last ticker overwrites the pawn's velocity readout - and a `CharacterMovement` keeps applying gravity against the flight dynamics. Opt out per-instance with **Pause Conflicting Movement Components**, or globally with `r.LcmNav.Flight.MovementAuthority 0`. Two notes: stock `MoveTo` path-following still binds the pawn's original movement component, so drive an enabled flight component with **FlyTo** (or `RequestVelocity`) - and the pawn's root primitive must not have **Simulate Physics** enabled (the component warns in the log if it does).

22.2.1 Stopping at the goal (Hold Station When Idle)

A movement component is told *what velocity to have*, so "stop" has always meant "target zero velocity" - and a body with momentum answers that by coasting until drag absorbs it, with nothing to bring it back. Measured settling error after arriving inside a 150 cm sphere and being told to stop: multirotor 267 cm past the goal, jet 936 cm, fish 2369 cm, and the bird 14,790 cm away and still descending.

Hold Station When Idle (on by default, `r.LcmNav.Flight.HoldStation`) makes an unguided component steer back to the point guidance handed it over at, on a gentle brake-shaped approach with a deadband

so a parked body is genuinely at rest. Any `RequestVelocity` call takes over immediately, so it only ever governs an otherwise-idle body. With it on, all four archetypes now settle within ~2 m of the goal and stay there.

22.3 Debugging the flight path - planned vs predicted vs flown

Turn on the navigation manager's **`bDrawPathfindingDebug`** (the master switch for all path debug) and a flight-component pawn draws three distinct lines:

line	what it shows	switch
Orange polyline + red spheres	the PLANNED path the planner delivered	manager `bDrawPathfindingDebug`
Magenta arc ahead of the pawn	the PREDICTED dynamic flight path - the component's real dynamics rolled forward along the remaining route (look-ahead, banking, braking included)	component `bDrawDebugPrediction` (default on) / `r.LcmNav.Flight.DebugPredict`, horizon `r.LcmNav.Flight.DebugPredictSec`
Green→red fading trail behind	the FLOWN trajectory (speed-coloured), plus yellow = velocity and cyan = guidance arrows	component `bDrawDebugTrail` (default off) / `r.LcmNav.Flight.DebugTrail`, lifetime `r.LcmNav.Flight.DebugTrailSec`

How to read them: the gap between orange and magenta is the body's *physics* (corner-cutting, brake distance - expected and correct); the gap between magenta and the trail is *live avoidance* (neighbours, walls - the prediction deliberately excludes what depends on other agents' futures). A lone agent whose trail leaves the magenta line is a tracking defect worth reporting. All three draw nothing in Shipping builds.

22.4 Archetypes (key `FlightConfig` params)

Archetype	Model	Notable params
Jet / Fixed-Wing (`Generic6DOF`)	Body-forward thrust, airspeed² lift carrying the weight, coordinated bank-to-turn, thrust-vectored VTOL blend below `WingBorneSpeed`	`MaxThrust`, `ThrottleResponse`, `WingBorneSpeed`, `LiftCoeff`, `MaxLiftFactor`, `BankTurnGain`, `MaxBankAngleDeg`, `VelocityP`, `AttitudeP/D`
Multirotor	Tilt-to-translate, altitude hold, spool lag, wind/gust, hover jitter	`RotorSpinUpRate`, `WindVelocity`, `Gust Strength/Frequency`, `HoverJitter Strength`, `bYawTracksVelocity`, `RotorMaxRPM`
Bird	Effort-throttled pulsed wingbeat, airspeed² lift, flap↔glide, banked turns, flare	`FlapFrequency`, `FlapThrust`, `LiftCoeff`, `MaxLiftFactor`, `BankTurnGain`, `MaxBankAngleDeg`, `FlareSpeed`, `FlareDragScale`
Fish	Neutral buoyancy, tail-beat undulatory thrust + sway, anisotropic hydro drag, current	`bNeutralBuoyancy`, `TailBeatFrequency`, `TailBeatFreqGain`, `TailWiggleStrength`, `LateralDragMult`, `WaterCurrent`

Shared: `MaxSpeed`, `MaxAcceleration`, `MaxTiltAngleDeg` (Multirotor), `MaxTurnRateDeg`, `SubSteps` (integration sub-steps; higher = stiffer + smoother), `BankingAmount`/`MaxBankingAngle` (kinematic-mode roll).

22.4.1 Winged bodies (Jet, Bird) - how they fly, and how to tune them

Both share one wing model, and it behaves like an aircraft rather than a hovering body:

- **Turns are made by banking.** The heading error commands a bank; the banked lift supplies the centripetal force; the nose then follows the *velocity*. So a winged body has a genuine minimum turn radius, `v² / (g·tan(MaxBankAngleDeg))` - 212 cm at 600 cm/s and 60°. Raise `MaxBankAngleDeg` (and `MaxLiftFactor` with it) for tighter turns; lower it for a heavy bomber.
- **Lift carries the weight, thrust only fights drag.** At cruise the throttle sits near `drag/MaxThrust` (~0.12 at defaults). `LiftCoeff` sets how much airspeed the wing needs: weight is balanced when `LiftCoeff ·v² = Mass·980`.
- **They cannot pinpoint-stop on aerodynamics alone.** Commanded to a hard stop, a wing has nothing to push against. The **Jet** answers this with a thrust-vectored blend below `WingBorneSpeed`; the **Bird** flares to shed speed and hover-flaps to hold. Both rely on the nav layer's arrival ramp - with a constant full-speed command any fast forward-flyer orbits its goal, which is expected.
- **Arriving is not the same as staying.** When a FlyTo task arrives it stops feeding guidance, and an unguided flight component **holds the position it was left at** rather than coasting to wherever momentum runs out (see *Hold Station When Idle* below). Without that a winged body sails past its goal and never comes back.
- **The Jet is `Generic6DOF` remodelled** (the enum name is unchanged so existing content keeps working). `r.LcmNav.Flight.JetGeneric 0` restores the previous hover-thruster behaviour exactly.

Notes:

- A fast forward-flyer (bird/fish) can't pinpoint-stop; the nav layer's **arrival slowdown** makes it flare to a perch. Without arrival-aware guidance a fast body orbits its goal (expected).
- Realistic dynamics *lag* guidance by design; raise `MaxAcceleration` for snappier, lower for heavier/driftier.
- **The two control loops are a cascade - tune them as a pair, never one alone.** `VelocityP` is the outer loop (time constant `1/VelocityP`); it commands a *tilt* that the inner `AttitudeP`/`AttitudeD` loop must then achieve. Keep the inner loop **~4× faster** than the outer one, or the body wanders side to side and hunts back and forth around its goal:

	formula	shipped default
inner natural frequency	`wn = sqrt(AttitudeP)`	7.75 rad/s
inner damping ratio	`z = (AttitudeD + AngularDrag) / (2*wn)`	0.84
inner time constant	`1/(z*wn)`	0.15 s
outer time constant	`1/VelocityP`	0.67 s

> ⚠ `AngularDrag` adds to `AttitudeD` - the integrator applies `-AttitudeD*w` and then `-AngularDrag*w`, so the loop's real damping is the **sum**. Raising `AttitudeP` for a snappier body without raising `AttitudeD` with it drops `z` below ~0.7 and the body visibly rings after every heading change. Damping must scale as `2*z*sqrt(AttitudeP)`.

22.4.2 Altitude while manoeuvring - `MaxTiltAngleDeg` and the thrust budget

A tilt-to-translate body (Generic6DOF / Multirotor) keeps only `cos(tilt)` of its thrust opposing gravity, so holding height while tilted needs a thrust-to-weight ratio of `1/cos(tilt)`:

tilt	lateral accel available	thrust-to-weight needed just to hold height
30°	566 cm/s²	1.15×
45°	980 cm/s²	1.41×
60° (default)	**1697 cm/s²**	**2.0×**
75°	3657 cm/s²	3.9×

`MaxTiltAngleDeg` caps this so the body cannot ask for a manoeuvre that costs it altitude, and when the thrust budget saturates the **vertical axis is served first** and the lateral axis takes the remainder. ⚠ Raising the tilt limit without raising `MaxThrust` to match makes the body descend under hard lateral commands - the harder you ask it to move, the faster it sinks. Keep `MaxThrust ≥ Mass * 980 / cos(MaxTiltAngleDeg)`; at the shipped 1 kg / 3000 / 60° there is 2× margin.

22.4.3 Reaching the commanded speed - drag is fed forward

The velocity loop cancels the drag it is about to apply, exactly as it already cancels gravity. That matters more than it sounds: drag is a *constant disturbance* to a proportional-only loop, and a P-only loop answers one with a **permanent offset** of `|drag| / (Mass * VelocityP)`. Left uncancelled at the shipped defaults that is 336 cm/s against a 600 cm/s command - the body would cruise at 44 % of what it was told, and no gain fixes it (raising `VelocityP` shrinks the error but costs the visible lean). Kill switch `r.LcmNav.Flight.DragFeedForward 0`.

22.4.4 Making a body ***lean*** (tilt-to-translate look)

A thrust-vector body leans because thrust must point along `Mass*accel + weight`. There is no cosmetic lean parameter in `FlightDynamics` mode - the attitude is physical, so you get lean by changing what the body has to *do*. Three separate mechanisms, commonly confused:

Want	Knob	Why
Forward lean while cruising	`LinearDragCoeff.X/Y` ↑	At steady speed accel≈0, so lean is exactly `atan(drag/weight)`. **No control gain can change this.** 0.5→1.4 takes it from 10°→28° at 600 cm/s.
Lean while accelerating/braking	`VelocityP` ↓	Peak lean is `atan(VelocityP*MaxSpeed/g)` but it decays over `1/VelocityP`, and the area is fixed. High gain = a spike too brief to see; low gain = a shallower lean actually held.
Bank into turns	`YawP` ↓ (keep ≥ `AttitudeP`)	The body banks only while its velocity error is *lateral*. Fast yaw whips the nose onto the new heading first, making the error forward instead - pivot-then-go, no bank.

⚠ In `Kinematic` mode none of the above applies: attitude there is cosmetic, driven by `BankingAmount` and `MaxBankingAngle`.

22.4.5 `BankingAmount` is now a multiple of the physically correct bank

`BankingAmount` (Kinematic mode, and the matching field on BT/StateTree FlyTo) is a **multiplier on the coordinated-turn angle** `tan(bank) = V * yawRate / g`. `1` is the true bank a real body holds in that turn; above 1 exaggerates it for readability; `0` flies wings-level.

⚠ **Its default changed from 4 to 1 and the old number does not carry over.** The previous law dotted a *normalised* per-frame velocity delta with the right vector - normalising throws away how hard the turn actually is, so the roll command sat near ±1 on ordinary velocity noise and the body flicked between ±4° every frame in straight flight. Multiplying the correct law by 4 instead saturates `MaxBankingAngle` in any ordinary turn. If you had authored a value here, re-tune it around 1. Legacy law: `r.LcmNav.Flight.CoordinatedBank 0`.

22.4.6 Flying straight up or down

Travel-referenced attitude is built by **parallel transport** of the previous frame, not from a world-up reference. This is not a refinement - the two obvious UE builders are *discontinuous* at vertical travel, and both were in use:

- `FVector::Rotation` yaws by `atan2(Y, X)`, which at `X = Y = 0` is `atan2(0,0) = 0`: a body climbing straight up snaps its heading to world +X, then swings with any lateral noise.

- `FRotationMatrix::MakeFromXZ` **substitutes** `(1,0,0)` for its up reference once `|Fwd.Z|` passes `1 - KINDA_SMALL_NUMBER` - the frame teleports rather than degrading.

Either one makes a climbing or diving agent roll hard for reasons invisible in its flight path. World-up levelling is still applied, weighted by how horizontal the travel is, so it fades out exactly where the reference stops being meaningful and the body simply holds its roll through vertical. Applies to Kinematic, Bird, Fish and both FlyTo tasks. Kill switch `r.LcmNav.Flight.ContinuousFrame 0`.

22.4.7 The vertical axis has its own gain - `VelocityPZ`

`VelocityP` is held low on purpose: horizontally it commands a **tilt** the attitude loop then has to achieve, so it must stay ~4× slower than that loop. The vertical axis has no such constraint - thrust acts on it **directly**, with no attitude change in between - so making it share the slow gain costs bandwidth for nothing. Every production multirotor controller splits these (PX4 `MPC_Z_VEL_P` vs `MPC_XY_VEL_P`; ArduPilot `PSC_VELZ_P` vs `PSC_VELXY_P`).

Measured on a closed-loop rig flying a **dead-level** path, so any altitude motion is controller-generated:

`VelocityPZ`	settled altitude wander	settled vertical speed
legacy (shares `VelocityP` = 1.5)	40.3 cm	33.4 cm/s
2	26.1	30.3
4 (default)	**6.0 cm**	**12.1 cm/s**
8	5.4	9.9

Cross-track error and arrival time are unchanged (173.8→173.6 cm, 10.85→10.83 s), so it costs nothing horizontally. **4 is derived, not chosen:** position-loop damping $\zeta = 0.5 \cdot \sqrt{(Kz \cdot L/v)}$ reaches 1.0 at Kz=4 when the look-ahead equals the speed, and the velocity loop's damping against the rotor spool, $\zeta = 1/(2\sqrt{(Kz \cdot \tau\_spool)})$, is 0.707 at exactly Kz=4 with `RotorSpinUpRate` 8. Raise `RotorSpinUpRate` too if you push it much past 4.

22.4.8 A path that ends short of the goal no longer orbits

If the delivered path's last waypoint is further than **2 × Acceptance Radius** from the goal, the task used to pin the index back to that waypoint *every tick* - and since the agent is already inside the acceptance radius of it, the next tick retired it and pinned again. Measured: **1750 pin events in 40 s**, with the task neither succeeding nor failing. It now pins once, holds position, replans twice, and then reports **Failed** with a log line naming the likely cause. Common causes of a short path: the goal quantises to a leaf centre, or `r.LcmNav.SeamSafe.TruncateUnprovable` (default ON) truncated the route at its first unprovable leg.

22.5 Troubleshooting - symptom to cause

Every row below is a *structural* cause with its own kill switch, so each can be A/B'd on its own. Work down the list; do not start by re-tuning gains, because none of these is a gain problem.

Symptom	Likely cause	Switch to A/B
Altitude swings up and down continuously, even in level flight	**The vertical axis shared the (deliberately slow) horizontal `VelocityP`, leaving the pure-pursuit position loop at ζ≈0.61 - underdamped, with nothing to damp it**	**`VelocityPZ` (default 4; set 0 for the legacy shared gain)**
Never reaches a waypoint, circles it forever, task never Succeeds or Fails	**End-of-path fallsafe re-pinned the index every tick when the path ended >2×AcceptanceRadius short of the goal**	fixed unconditionally; look for the `[FlyTo] giving up:` warning

Symptom	Likely cause	Switch to A/B
Altitude rings up then sags whenever it starts a horizontal move	Thrust magnitude applied along the body's *current* axis instead of projected onto it	`r.LcmNav.Flight.Thrust Projection`
Sinks during hard turns or fast starts	Tilt unbounded / thrust saturation cutting the vertical component	`r.LcmNav.Flight.Thrust Priority`, then raise `MaxThrust` or lower `MaxTiltAngleDeg`
Never reaches its commanded speed; feels unresponsive	Drag not fed forward → permanent P-loop offset	`r.LcmNav.Flight.DragFeed Forward`
Rolls violently when climbing or diving steeply	Discontinuous world-up attitude reference	`r.LcmNav.Flight. ContinuousFrame`
Twitches side to side in straight, level flight	Roll commanded from a *normalised* velocity delta	`r.LcmNav.Flight. CoordinatedBank`
Lurches when it passes a waypoint or an avoidance term flips	Discontinuous guidance command stepping the cascade	`r.LcmNav.Flight.Guidance Slew`
Judders / grinds along a surface it is touching	Velocity not corrected after a blocking sweep	`r.LcmNav.Flight.Slide Velocity`
Weaves near geometry it is not even approaching	Wall whiskers cast off the *nose* (which lags travel), binary, no forward or vertical coverage	`r.LcmNav.WallAvoidV2 1` (⚠ default 0 - see below)
Bird's wingbeat stutters on and off at cruise	Flap/glide threshold had no hysteresis	fixed unconditionally (Schmitt trigger on `GaitEffort`)
Fish rolls in time with its own tail beat	Wiggle force fed back into the attitude error	fixed unconditionally
Passes waypoints / overshoots the goal	Guidance solving stopping distance against `MaxAcceleration`, a ceiling the loop rarely reaches	`r.LcmNav.FlightGuidance AccelFromVelocityP` (already default 1)
Weaves along a dense path	Pure-pursuit look-ahead shorter than the body's response distance	`r.LcmNav.PursuitLagFloorK` (already default 1.5)

⚠ **`r.LcmNav.WallAvoidV2` defaults to 0 on purpose.** Reducing this term's authority has been *measured harmful* once already - a bounded version cost a single agent its arrival (33 loop events, orbiting), because a planned route sometimes points into a wall and only a full-authority push clears it. V2 keeps full authority at contact and only ramps the far field, but that measurement has not been retaken against it. Turn it on if flyers judder near geometry, and re-check arrival rates.

22.6 Animation contract - `FLcmFlightState`

Every tick the component produces `FLcmFlightState` (archetype-agnostic where possible):

Field	Meaning
`Speed01`	Normalized airspeed (|velocity| / MaxSpeed) - blendspace axis
`VerticalSpeed`	cm/s, climb + / dive − - climb/dive pose blend
`BankAngleDeg` / `PitchDeg`	Body roll / pitch - additive lean/pitch
`YawRateDeg`	Turn rate - lean/tilt into turns
`GaitPhase`	0.1 cyclic driver: rotor spin / wingbeat / tail-beat

Field	Meaning
`GaitAmplitude01`	Throttle (drone) / flap-glide blend (bird) / undulation vigour (fish)
`RotorRPM`	Multirotor rotor RPM
`Throttle01` / `Airspeed`	Commanded effort / raw speed

Consume it two ways:

- **Pull:** `GetFlightState` (BlueprintPure) on the component, e.g. from an AnimBP's Update.
- **Push:** implement **`ILcmFlightAnimInterface::ReceiveFlightState`** on your AnimBP (or pawn). The component calls it every tick (`bBroadcastAnimState`, default on). Cache the struct and drive:
  - `GaitPhase` → rotor/wing/tail cyclic pose (Sine/curve on the phase)
  - `Speed01` + `VerticalSpeed` → locomotion blendspace
  - `BankAngleDeg` / `PitchDeg` → additive lean/pitch
  - `GaitAmplitude01` → flap-vs-glide (bird) or effort (fish/drone) blend weight
  - `RotorRPM` → prop-spin material/WPO param (Multirotor)

22.7 Mass crowds & scale (P7-P8)

The **LCM Nav3D Agent** trait (NavMode = PathFollowing) carries the same flight options (`LocomotionModel`, `FlightArchetype`, `FlightConfig`) - set them and a whole crowd flies with the identical 6-DOF physics as a pawn, still pathing + avoiding + arriving. Default `Kinematic` → byte-identical to before.

Dynamics LOD (the scale knob). Full 6-DOF per agent is expensive; set `FlightDynamicsLODDistance` (cm) so only agents within that distance of the viewer run full dynamics - everyone else uses the cheap kinematic integrator (face-velocity, no roll). So a 10K crowd keeps only the on-screen agents on the expensive path. `0` = always full. Global kill switch: `r.LcmNav.Mass.FlightLOD` (1 default / 0 = full everywhere). LOD switching is seamless - a far agent's integrator state re-syncs from its live transform when it returns to range.

Deferred (documented) scale extensions:

- **GPU 6-DOF compute** for very large crowds is a designed extension, intentionally not shipped here: CPU dynamics-LOD already bounds cost to the visible set, which is the cheaper win. Revisit with engine uplift.
- **Shared flight config** - `FlightConfig` is currently per-entity; moving it to a const-shared fragment per archetype would cut crowd memory. Deferred (shared-fragment archetype plumbing).

22.8 Determinism

The dynamics are deterministic: fixed integration sub-steps, no RNG or wall-clock; gust/jitter are seeded per-agent sinusoidal noise. Same input sequence → identical trajectory.

22.9 Bird and Fish do not share the drone control cascade

⚠ **`StepBird` and `StepFish` are separate integrators - they never call `RunStep6DOF`.** That is deliberate (a bird does not translate sideways by tilting a rotor), but it meant they also never had its velocity-tracking loop: thrust acts along body-forward only, and until 2026-08-06 **nothing corrected a velocity error perpendicular to the nose**. They pointed at the guidance direction and waited for drag, so cross-track and altitude error could only decay passively - never be driven to zero. That is what "drifts away from the path points" was.

They now generate the perpendicular force a banked wing or a flexing body actually produces, bounded by what the body can physically supply - **lift** for the bird, **tail/fin thrust scaled by swim effort** for the fish. `BankTurnGain` is the rate (1/s) that converts a cm/s error into that demand, which is what its own description always claimed. Measured on one corner path, same guidance:

archetype	cross-track before → after	altitude error before → after
Generic6DOF (reference)	174.3 → 174.3	9.4 → 9.4
Multirotor (reference)	173.6 → 173.6	22.8 → 22.8
Bird	**234.4 → 183.9**	69.7 → 61.5
Fish	203.7 → 198.4	**64.0 → 17.4**

Both drone archetypes are byte-identical (their gate checksums are unchanged), because the change is scoped to the two integrators that lacked the loop. Kill switch `r.LcmNav.Flight.BodyManeuver 0`.

⛔ **Do not "simplify" the bird by removing its `LiftControl` trim as a duplicate vertical controller.** Measured and reverted: it is the bird's only means of balancing lift against gravity - lift is `LiftCoeff × airspeed²`, which at cruise exceeds body weight, so without the trim the bird climbs away (altitude error 61.5 → 186.4 cm). The trim sets lift *magnitude* for equilibrium; the manoeuvre force supplies *perpendicular authority*. They are not redundant.

22.10 Hard turns: progress does not depend on hitting a sphere

⚠ **If a tighter Acceptance Radius makes your agents stick, this is why.** A waypoint used to be retired only by `HasReachedOrPassedWaypoint`: *within Acceptance Radius*, **or** *past the plane through it normal to the incoming leg*. At a corner sharper than the body's minimum turn radius - `v / VelocityP`, which is **400 cm** at the shipped defaults - the agent cannot pass within a 100 cm radius, and the plane has rotated with the path so it is not past that either. **The gate becomes unsatisfiable**, the index stalls on a point now *behind* the agent, and because pure pursuit measures its look-ahead *from the current index*, the aim point sits behind too and drags the agent backwards.

Measured on 375 cm waypoint spacing (what `ChunkSize` 3000 produces), 135° corner:

acceptance radius	gate only	+ projection advance
50 cm	**stuck** - 39.9 s stalled, 1214 backtrack ticks	arrives, 0 backtrack
100 cm	**stuck** - 40.0 s stalled, 1214 backtrack ticks	arrives, 0 backtrack
200 cm	arrives	identical
400 cm	arrives	identical

The index now *also* advances by **projecting the agent onto its path**, so progress is monotone and independent of the acceptance gate. It is **inert** wherever the gate already works (45° and 90° corners are byte-identical; 200 cm and 400 cm radii are byte-identical). On the demo map at `ChunkSize` 3000 with 7 agents it took arrivals from 1/7 to 3-4/7 and mean cross-track from 392 cm to ~225 cm.

⚠ **`r.LcmNav.PathProjectWindow` (default 3) is load-bearing, not an optimisation.** On a hairpin the return leg runs spatially alongside the outbound one, so an unbounded nearest-segment search would snap the agent onto the return leg before it had rounded the corner and cut the whole turn. Kill switch for the whole behaviour: `r.LcmNav.PathProjectAdvance 0`.

⚠ Acceptance Radius still does not control path adherence - it never did. It decides when a point is *retired*, not how closely the agent flies. Corner-cutting is bounded by physics: minimum turn radius is `v²/a_eff`, so waypoints spaced closer than that will be passed wide however you tune this.

22.10.1 Scope: every follower rework in this section is 6-DOF ONLY

⛔ The behaviours documented below - progress-based stall detection, the visibility-clamped look-ahead, the backward speed profile, and the projection index advance - apply **only** to a pawn driven by an enabled `ULcmFlightMovementComponent` in **FlightDynamics (6-DOF)** mode.

The **Kinematic (legacy point-mass)** model and pawns with no component at all (which the task integrates itself) keep their original behaviour exactly: the acceptance-gate-only waypoint advance, the unclamped

look-ahead target, the one-corner speed cap, and the displacement-based stall timer. Those are separate shipped locomotion models whose feel users have already built content against.

⚠ **Both defaults point away from 6-DOF** (`LocomotionModel = Kinematic`, `bUseFlightMovementComponent = false`), so a **default-configured pawn gets none of this**. It is opt-in, by design. `LcmFlightLocomotion::IsSixDofBody` is the single predicate; do not re-derive it.

The same rule covers the attitude rework (`r.LcmNav.Flight.LegacyAttitudeOutside6DOF`, default 1) and the component's guidance-reference limiter and slide-velocity correction, which are likewise 6-DOF only. The limiter in particular **must not** reach Kinematic: `ELcmFlightStyle::LinearDirect` exists to answer a command directly and `PhysicsWeighted` to answer it with inertia, and handing `LinearDirect` a reference already ramped at `MaxAcceleration` makes the two indistinguishable.

22.10.2 Stall recovery measures PROGRESS, not motion

An agent that is flying in a circle is, to every speed-based health check, perfectly healthy. That is not a hypothetical: on `Vertical_Skyscraper` at `ChunkSize` 3000 the flyer went **828 m to gain 35 m** (tortuosity 23.8, 53 loop events) and *every* recovery counter read zero - `stallFrac` 0.00, `escapeTicks` 0, `replans` 0, `paths` 1. It orbited the first storey until the run timed out.

The cause was one line. The stall timer reset whenever the pawn had moved more than 50 cm from `StallEscapeAnchor` - and then moved the anchor to the pawn:

```
if (FVector::DistSquared(NowPos, StallEscapeAnchor) > 2500.0f) // ">50 cm = progress"
{ StallEscapeAnchor = NowPos; StallEscapeSeconds = 0.0f; }
```

An orbiting agent clears 50 cm every few frames, so the timer could never accumulate. **The one recovery path that could have broken the loop was held open by the loop itself.**

Progress is now **remaining arc length along the path** (`ULcmSteeringMath::RemainingPathLengthCm`), which falls monotonically along any correct traversal and does not fall at all while orbiting.

⛔ **Straight-line distance to the goal is the wrong scalar and must not be substituted.** A correct route routinely moves *away* from the goal - the horizontal leg of a vertical switchback, or any detour round an obstacle - so a goal-distance watchdog fires on healthy navigation.

⚠ `StallBestRemainCm` **must** be reset to `TNumericLimits<float>::Max` every time `CurrentPath` is replaced. A fresh path legitimately has a larger remaining length; without the reset it reads as permanent no-progress and arms recovery forever.

`Vertical_Skyscraper`, CS 3000	legacy	progress-based
outcome	stuck at Z=−6999 (first storey)	climbs the tower
closest approach	15855 cm	600 cm (143 cm with a shorter look-ahead)
tortuosity	23.8-31.6	5.6-6.8
loop events	53-68	0-13
replans	1	19-39

The legacy failure is bit-reproducible (15855 cm / 31.56 / 68 loops / 1 path on every run), which makes this map a far better regression instrument than the 7-agent demo scenario.

Kill switch: `r.LcmNav.ProgressStallDetect 0`. Threshold: `r.LcmNav.ProgressEpsilonCm` (default 50).

22.10.3 The look-ahead target must be REACHABLE, not merely on the path

Pure pursuit measures its look-ahead **along the path**, then flies at the result in a **straight line**. Those are the same thing only where the path is locally straight. In a vertical shaft with offset floor openings, a target 400-1000 cm along the path sits on the far side of a **slab**, and the straight line to it goes through solid.

The agent charges a point it cannot reach, is blocked, retries, and orbits - the user-visible *"it reaches for a further point before finishing the nearest one, then gets stuck"*.

`ULcmSteeringMath::GetPursuitTargetVisible` walks the look-ahead back until the agent→target segment is clear in the SVO, and falls back to `Path[CurrentIndex]` - the nearest unconsumed waypoint, which is always a safe aim because the planner already proved the path traversable. At most 4 line-of-sight tests per tick.

⚠ **It fails OPEN.** With no chunk data - unstreamed, or the segment leaves the single chunk this test can see - it returns the unmodified target, i.e. exactly legacy behaviour. Never infer a blockage from missing data; that would brake the follower at every chunk boundary.

`Vertical_Skyscraper`, CS 3000, progress detector on	look-ahead unclamped	**visibility-clamped**
arrives	no, no	**yes, yes**
time to goal	never	**80.12 s, 80.25 s**
closest approach	6615, 6615 cm	**144, 145 cm**
loop events	46, 46	**0, 0**
replans	40, 40	15, 15

Both fixes are needed. The progress detector alone breaks the orbit-lock but still ends 66 m short; the visibility clamp alone leaves the stall detector blind to any orbit it does not prevent.

Kill switch: `r.LcmNav.PursuitVisibleTarget 0`.

Regression on `Mixed_Indoor_Outdoor` (same pinned single-agent instrument, 2 runs per arm): loop events fell **48/46 → 3/1** there too, with `trackErrMax` unchanged (2326/2406 → 2683/2133, overlapping). ⚠ Replans rose 23/10 → 76/64: that is the progress detector doing its job - the agent now *notices* it is getting nowhere instead of orbiting silently - but it is real planner load. Raise `r.LcmNav.ProgressEpsilonCm` if that rate is too high for your budget.

⚠ **`ChunkSize` 8000 cannot navigate this map at all, for an unrelated reason.** It yields leaves up to 1000 cm, too coarse to resolve the floor openings, so the slabs seal the shaft into disconnected components and no path exists to plan (`portalsInAgentComp=0`, `open set exhausted`). The agent never moves because there is nothing to follow - do not read that as a follower defect.

22.10.4 Speed planning: the agent slows down *for corners it can already see*

Fixing the index advance stops the agent sticking, but it does not stop it going into a corner **too fast**. Those are separate defects with the same symptom, and the second one is speed-dependent - which is exactly the "fine when it's slow, sticks when I speed it up" report.

The speed governor's corner cap used to look **exactly one waypoint ahead**: it capped speed at `sqrt(v_corner² + 2·a·d)` where `d` is the distance to the *single* next waypoint. That is the right equation solved over the wrong distance. Braking distance grows as $\mathbf{v^2}$, waypoint spacing is fixed by the voxel grid, and the body's response lag is a fixed `v / VelocityP` - so past some speed the brake command is issued too late to be achievable, and the agent simply carries its speed through the corner. Measured at a 135° corner on 375 cm spacing, `MaxSpeed` 1200: the old cap held mean speed to 823 cm/s - *precisely* `sqrt(v_corner² + 2·900·375)`, i.e. it did exactly what it was written to do, and that was not enough. Peak deviation from the path was still 371 cm. In a corridor, 371 cm is a wall.

The governor now builds a proper **backward velocity profile** over a window (`r.LcmNav.SpeedProfilePoints`, default 8):

1. Give every upcoming waypoint the speed *its own* curvature allows, `v_i = sqrt(a · R_i)`.
2. Sweep **backward**: `v_i = min(v_i, sqrt(v_{i+1}² + 2·a·d_i))`, so each corner's limit propagates upstream as far as the braking distance actually needs.
3. Brake from the agent's position to **every** limit in the window and obey the tightest, charging the body's response distance `v / VelocityP` against each available braking distance.

Peak cross-track at a 135° corner, 375 cm spacing, by `MaxSpeed`:

`MaxSpeed`	ungoverned	one-corner cap (old)	**backward profile**	mean speed, old → new
300	94.5 cm	94.5 cm	95.0 cm	284 → 281
600	201.5 cm	184.8 cm	**93.4 cm**	517 → 506
900	287.2 cm	268.1 cm	**89.8 cm**	693 → 688
1200	397.8 cm	371.4 cm	**103.5 cm**	823 → **827**

Tracking error is now **flat in speed** instead of growing linearly with it, and it is not bought with travel time - mean speed is unchanged (at 1200 it is marginally *higher*, because the agent stops holding a compromise speed down the straight and instead runs fast then brakes late-but-sufficiently).

⛔ `r.LcmNav.SpeedProfilePoints 1` narrows the window to a single corner but is **NOT** the legacy cap, and must not be used as a way to restore it. The old cap took the incoming leg from the agent's own position (`Path[i] - CurrentLoc`); the profile takes it from the path (`Path[i] - Path[i-1]`), because path geometry is the only definition available for the waypoints further out. The braking distances differ, and on a `LinearDirect` body that difference shows up as overshooting the waypoint. Non-6-DOF bodies therefore run the original block verbatim rather than a 1-wide profile.

On the demo map (`Mixed_Indoor_Outdoor`, `BT_FlightBehavior_Loop`, single agent, pinned start/goal, 3 runs per arm) exactly one follower metric moves, and it is the one that matters:

metric	one-corner cap	backward profile	
worst-case cross-track	3570 / 3244 / 3282 cm	**2536 / 2560 / 2544 cm**	−24%, no overlap
mean cross-track	1226 / 1197 / 1320	839 / 1232 / 1047	overlaps - no effect
commanded deflection	13.7 / 7.7 / 2.8°	1.8 / 5.0 / 16.1°	overlaps - no effect
avoidance-dominated ticks	10 / 7 / 4 %	2 / 4 / 10 %	overlaps - no effect

The worst-case figure is not merely better, it is **tight**: ±12 cm across runs whose replan counts ranged 16-47, against ±170 cm for the one-corner cap. That is the signature of a bound that now holds structurally rather than by luck. *Mean* cross-track not moving is consistent with the mechanism - the profile acts at corners, and the mean is dominated by the straights, where both arms are identical.

⚠ **Do not measure this with the 7-agent configuration.** ORCA/flocking interference, a looping BT that disperses agents differently each run, and an unpinned start together produce a **57× spread within a single arm** (wall-avoidance term 0.08 to 4.60), which is far larger than the effect. Arrival count alone came back 0, 1 and 3 of 7 across runs of the *same* build. Pin the start and goal and use one agent. Note also that `reach=` is meaningless under a looping BT - it cycles targets by design.

⛔ **Do not "improve" step 3 by walking the response distance forward along the path and evaluating the profile at that single lagged point.** It reads as the more principled model of a lagging body, and it is worse at every speed (166 / 238 / 288 cm against 95 / 163 / 247): once the lagged point passes the corner, the limit there is the *post*-corner straight-leg limit, so the governor releases the brake while the body is still entering the turn. The lag credit is sound for accelerating and unsound for braking - only the pessimistic direction may use it.

CHAPTER 23

LCM Nav3D - Mass Entity Trait Reference

How to build a Mass crowd/agent with LCM Nav3D. Add these traits to a **Mass Entity Config asset** (`Create → Miscellaneous → Mass Entity Config`) and use it on an `AMassSpawner`.

> **One rule:** every agent needs **exactly one movement trait**. Everything else is an orthogonal add-on. The editor validator (below) enforces this.

23.1 The anatomy of a config

A Mass Entity Config is a list of traits. A working Nav3D agent needs one from each of three categories, and no more than one from the movement category.

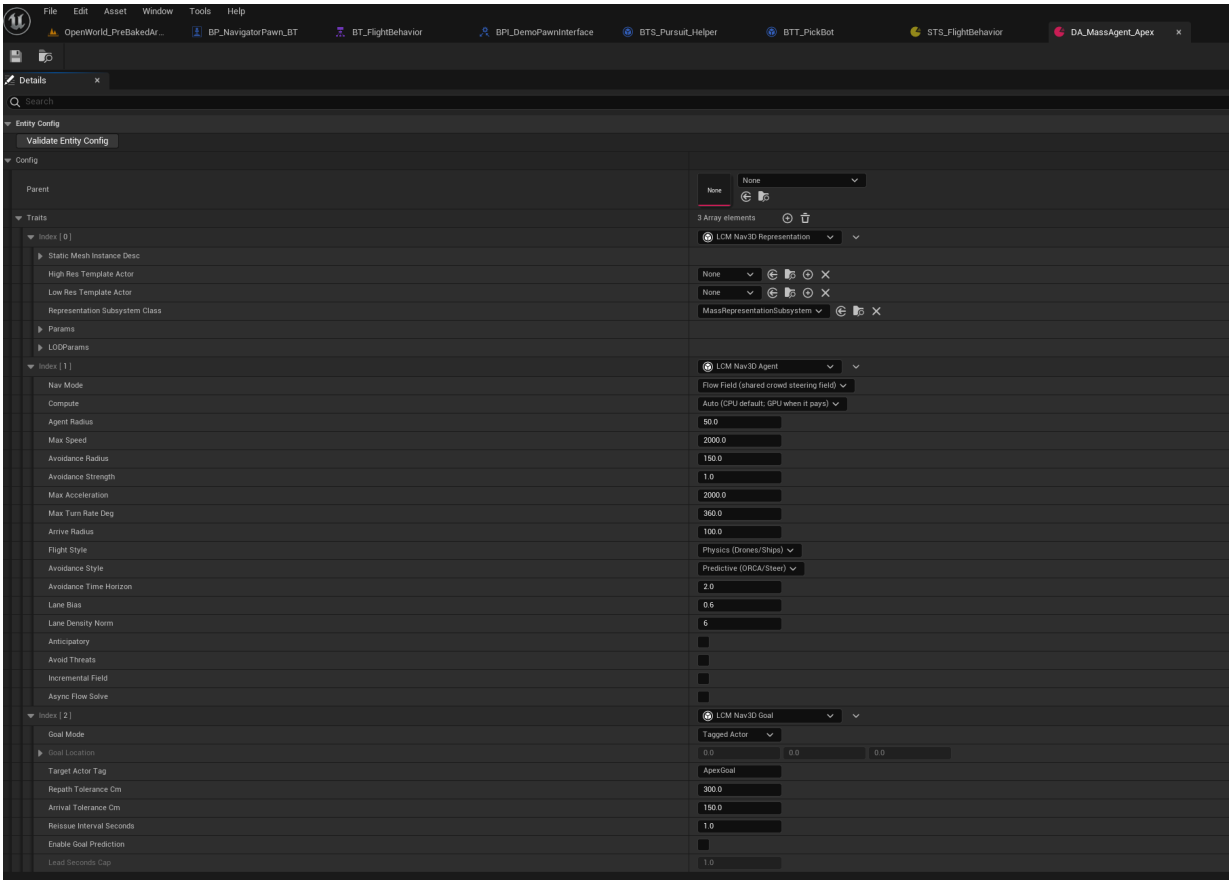


Figure 1 - A three-trait config, one per category.

Index	Trait	Category	Why it is here
`[0]`	**LCM Nav3D Representation**	Rendering - *pick one*	Without a rendering trait your agents move correctly and are **invisible**

Index	Trait	Category	Why it is here
`[1]`	**LCM Nav3D Agent**	Movement - ***exactly one***	Its `Nav Mode` picks Path Following or Flow Field
`[2]`	**LCM Nav3D Goal**	Goal	Here `Goal Mode = Tagged Actor`, targeting tag `ApexGoal`

Things the screenshot shows that are easy to miss:

- **`Validate Entity Config`** at the top runs the trait-combination check on demand. It also runs on save and on cook.
- **`Compute`** - leave it on `Auto`. This release solves on the CPU, so `Auto` and `CPU` behave identically; the selector is kept for forward compatibility.
- **Greyed rows are not disabled features.** `Goal Location` is greyed because `Goal Mode` is `Tagged Actor`, and `Lead Seconds Cap` because `Enable Goal Prediction` is off. The panel hides what the current configuration will ignore.

23.1.1 ⛔ This config is also the worked example of the one rule

Look at `[1]` and `[2]` together. The Agent's **`Nav Mode` is `Flow Field`**, but **`LCM Nav3D Goal` is a Path Following add-on** - it drives the *path-request* fragment, and a Flow Field agent never issues one. So this combination **fails validation**, with:

> *LCM Nav3D Goal only works with a Path Following agent (it drives the path-request fragment). For a Flow Field agent, set the goal on the 'LCM Nav3D Agent' trait's Goal / Goal Mode properties instead.*

The fix is one of two things, and which one depends on what you want:

You want	Change
A 10K-scale crowd sharing a goal	**Remove the Goal trait.** Set the goal on the Agent trait's own `Goal Mode` properties.
Per-agent precise routes	**Set `Nav Mode` to `Path Following`.** The Goal trait is then correct as it stands.

Press **Validate Entity Config** on any config you build. It catches this, the duplicate-movement-trait case, and the mixed Path+Flow case - all three of which are silent at runtime and present as "my agents do not move".

23.2 The traits

23.2.1 □ `LCM Nav3D Agent` - the one movement trait (start here)

The single authoring entry point. Pick a **Navigation Mode**:

Navigation Mode	What it does	Fragments composed
Path Following	Per-agent A* through the SVO (HPA* across chunks). Best for hero/mid-count agents, precise routes, infinite worlds.	`NavAgent`, `PathRequest`, `PathResult`

Navigation Mode	What it does	Fragments composed
Flow Field	One shared Dijkstra flow field per goal, O(1)/agent sampling. Best for 10K-scale crowds sharing goals (finite worlds).	`FlowField` (state) + `FlowConfig Shared` (const shared config)

Mode-specific properties hide automatically via `EditCondition`. Common movement knobs (radius, speed, accel, turn-rate, avoidance style, flight style, lane bias) apply to both.

23.2.2 + `LCM Nav3D Goal` - auto-pathing (Path Following only)

Points a **Path Following** agent at a fixed location or a tagged actor and keeps it pathed there (re-pathing on drift, optional lead-prediction for moving targets). **Not for Flow Field** - set the goal on the Agent trait's *Goal Mode* properties instead. The validator warns if you get this wrong.

23.2.3 + `LCM Nav3D Perception` - SVO-LOS sight (either mode)

Makes an agent both a *senser* and *perceivable* (faction-based, true-3D SVO line-of-sight). Writes a per-agent "closest visible threat" surface read by EQS, StateTree, or steering. Opt-in avoidance: `r.LcmNav.Mass.PerceptionAvoid 1` makes Flow Field agents veer from their own perceived threat.

23.2.4 □ Rendering - pick ONE

Trait	Use when	LOD?	Animation
LCM Nav3D ISM Render	Turnkey, hundreds-low thousands, "just draw a mesh".	No	Yes - 2 per-instance custom-data floats: `[0]` speed01 (walk/run blend), `[1]` stable phase (de-sync). Feed a VAT/WPO material.
LCM Nav3D Representation	LOD-culled crowds; optional hero actor up close. Wraps Epic MassRepresentation with sane defaults (draws ISM at all visible LODs, culls far). **Needs a viewer/player in the level** for LOD distance.	Yes	Via the mesh material / actor
(Niagara GPU crowd)	10K+ (deferred / roadmap)	-	VAT

23.3 Combination matrix

	Agent (Path)	Agent (Flow)	Goal	Perception	ISM Render	Representation
Agent (Path)	-	✗ conflict	✓	✓	✓	✓
Agent (Flow)	✗ conflict	-	⚠ use Agent goal	✓	✓	✓
Goal	✓	⚠ redundant	-	✓	✓	✓
Perception	✓	✓	✓	-	✓	✓
ISM Render	✓	✓	✓	✓	-	✗ pick one

	Agent (Path)	Agent (Flow)	Goal	Perception	ISM Render	Representation
Representation	✓	✓	✓	✓	✗ pick one	-

- ✗ **conflict** - two movement stacks; the editor validator errors and (at runtime) the two steerers mutually exclude → the entity would not move. Keep one.
- ⚠ - allowed but pointless; the validator explains.

23.4 The editor validator

`ULcmMassConfigValidator` runs on every `UMassEntityConfigAsset` (right-click → Validate, or on save/cook). It errors on:

1. **Two movement stacks** (Path + Flow, in any mix of unified/legacy traits).
2. **Goal without a Path Following agent**.

So a broken config can't ship. Requires the **Data Validation** plugin (bundled).

23.5 Migration from the old traits

The pre-2026-07 traits still work (byte-identical archetypes) but are marked **(Legacy)** and superseded by the unified trait:

Legacy trait	Replace with
`LCM Nav3D Agent (Legacy)`	`LCM Nav3D Agent`, NavMode = **Path Following**
`LCM Nav3D Flow Steer (Legacy)`	`LCM Nav3D Agent`, NavMode = **Flow Field**

Existing content (e.g. `DA_MassAgent_Apex`) keeps working - no forced migration.

Default-value deltas when migrating (the unified trait has ONE default per property; the two legacy traits differed): `ArriveRadius` is now **100** (legacy path 80 / legacy flow 150) and `AvoidanceStyle` defaults to **Reactive** (the legacy Flow Steer defaulted to Predictive/ORCA). If your config relied on a legacy default, set the value explicitly after switching traits. `TargetActorTag` defaults to `ApexGoal` (matches legacy).

23.6 World Partition / large-world (City Sample scale) - REQUIRED READING

23.6.1 The vanishing-crowd trap (stock engine behavior)

`AMassSpawner::BeginDestroy` calls `DoDespawning` - **the spawner destroys every entity it spawned when the spawner actor is destroyed** (engine `MassSpawner.cpp`). In a World Partition map a placed spawner is **spatially loaded by default**, so when the player leaves the spawner's cell, the spawner streams out and **silently deletes the entire crowd**. Symptom: agents work near the spawn area, then all disappear when the player walks away.

Fix (pick one):

1. Use **`LCM Nav3D Crowd Spawner`** instead of the stock Mass Spawner - identical (EQS generators etc. all inherited) but defaults to *not* spatially loaded, so its crowd's lifetime is never tied to the player's position.
2. On a stock `AMassSpawner`: Details → World Partition → **Is Spatially Loaded = false**.

This follows the AAA rule: **population lifetime must be owned by something persistent** (an always-loaded actor/subsystem), never by a streamed actor. Distance-based *despawning* should be an explicit population policy (an ambient ring, à la City Sample's crowd), not a streaming side effect.

23.6.2 City-scale checklist

- **Navigation**: a city does not fit the finite default (`WorldExtent` = 100 m). Use the manager's **Infinite World** mode (Hierarchical HPA*); the flow crowd keeps its own nav chunks resident via crowd-driven streaming (`r.LcmNav.Mass.CrowdStreaming`, default on) - agents don't lose nav when the player (the only invoker) leaves.
- **Rendering**: `LCM Nav3D Representation` culls agents *visually* past its LOD distances (defaults: Off at 200-250 m - deliberately conservative). For city vistas, raise `LODParams.BaseLODDistance/VisibleLODDistance` on the trait. This is visual LOD only - entities keep simulating; it is NOT the vanishing-crowd trap above.
- **Ambient population ring** (constant-density crowds around the player with bounded cost, the City Sample pattern) is a roadmap feature - until then, crowds are persistent: they exist wherever you spawned them.

23.7 Useful CVars

CVar	Default	Effect
`r.LcmNav.Mass.PerceptionAvoid`	0	1 = Flow agents veer from their perceived closest threat.
`r.LcmNav.Mass.PerceptionAvoidRadiusCm` / `...Strength`	600 / 1.0	Perception-avoidance range + strength.
`r.LcmNav.Mass.ISMAnimRefSpeedCmS`	600	Speed (cm/s) mapping to ISM custom-data `speed01 = 1`.
`r.LcmNav.Mass.Anticipatory`	−1 (per-trait)	Route Flow crowds around where obstacles are *heading*.

Part V

Appendix

CHAPTER 24

LCM Nav3D - API Reference

The classes, functions and properties you can reach from Blueprint, the Details panel, a Behaviour Tree, a StateTree or a Mass config.

> Internal C++ - solver internals, replication contracts, subsystems with no Blueprint surface - is deliberately not listed. If a type is not here, you are not expected to call it.

24.1 Classes

24.1.1 `LcmAIPerceptionBridge`

Mirrors UAIPerceptionComponent-sensed actors into a decaying threat tracker for the tactical EQS surface.

Functions

- `HandleTargetPerceptionUpdated` - Bound to the paired UAIPerceptionComponent's OnTargetPerceptionUpdated.
- `GetTrackedThreats` - Current tracked-threat actors (post-decay).

Properties

- `DecayWindowSeconds` - Seconds a lost-sight threat stays queryable before it decays out.
- `PerceptionComp` - The paired perception component (found on the owner at BeginPlay).

24.1.2 `LcmBTDecorator_ChunkLoaded`

Decorator that passes only when the SVO chunk containing the queried world position is currently loaded (cheap single-endpoint hash lookup, no path probe).

Properties

- `QueryLocationKey` - Blackboard key holding the world position to probe. Accepts Vector or Object (Actor location is used in the latter case).
- `bRequireFullResolution` - When true (default), only fully-resolved chunks (post-T12 promotion) satisfy. When false, any loaded chunk does.

24.1.3 `LcmBTDecorator_HasLineOfSight`

Decorator that passes when the agent has volumetric line of sight to the target via the Lazy Theta* LOS walker, honouring AgentRadius through the JFA clearance gate.

Properties

- `TargetKey` - Blackboard key for the LOS target. Vector = world location, Object = AActor (its current location is used).
- `AgentRadius` - Agent collision radius for the JFA clearance gate. Higher = more conservative (rejects narrow gaps).

24.1.4 LcmBTDecorator_IsInCover

Decorator that passes when the agent is in cover relative to a threat: its leaf SweepPenalty is at or below MaxSweepPenalty AND the threat's line of sight to it is blocked.

Properties

- `ThreatKey` - Threat key. Vector = world location, Object = AActor.
- `MaxSweepPenalty` - Highest SweepPenalty (0.255) the agent's leaf may carry to count as safe. Default 32 ≈ "lightly swept, mostly safe".
- `AgentRadius` - Agent collision radius for the LOS walker's clearance gate.

24.1.5 LcmBTDecorator_PathLengthBelow

Decorator that passes when the true-3D LCM Nav3D path distance from the agent to the goal is below MaxLengthCm (uses the synchronous tactical-library path probe; do not run per-frame).

Properties

- `GoalKey` - Blackboard key for the goal endpoint. Vector = world location, Object = AActor (its current location is used).
- `MaxLengthCm` - Maximum allowed path length, in centimetres, for the decorator to pass.
- `AgentRadius` - Agent collision radius (cm) used by the path probe's clearance gate.

24.1.6 LcmBTDecorator_ThreatVisibleViaPerception

Decorator that passes when AIPerception currently perceives a hostile AND the agent has clear volumetric LOS to it (the "can-shoot-now" gate).

Properties

- `ThreatKey` - Optional explicit threat key. When unset (NAME_None), the decorator uses AIPerception's closest currently-perceived actor.
- `AgentRadius` - Agent collision radius (cm) used by the LOS walker's clearance gate.

24.1.7 LcmBTService_NavigationLoad

Service that each tick writes the navigation manager's SVO streaming stats into a blackboard key so designers can gate expensive LCM Nav3D queries on idle streaming.

Properties

- `LoadedChunkCountKey` - Int blackboard key receiving the count of currently loaded SVO chunks.

24.1.8 LcmBTService_PathProgressMonitor

Service that each tick writes a Euclidean approach-progress fraction (1 - dist/initial_dist) and remaining distance to the goal into blackboard keys.

Properties

- `GoalKey` - Blackboard key for the goal. Vector = world location, Object = AActor (its current location is used).
- `ProgressKey` - Float blackboard key receiving the progress fraction (0.1) toward the goal.
- `RemainingCmKey` - Float blackboard key receiving the remaining straight-line distance (cm) to the goal.
- `GoalChangeTolerance` - Distance (cm) the goal must move before the progress baseline is re-anchored.

24.1.9 LcmBTService_SquadSnapshot

Service that each tick aggregates perceived allies within SquadRadius and writes ally count, squad centroid, and cohesion radius into blackboard keys.

Properties

- `AllyTag` - Tag filter - only perceived actors with this tag count as ally.

- `SquadRadius` - Radius (cm) around the agent within which allies are counted.
- `bIncludeSelf` - When true, the agent itself is included in the squad aggregate.
- `NumAlliesKey` - Int blackboard key receiving the number of allies in the squad.
- `CentroidKey` - Vector blackboard key receiving the squad's centroid (average ally position).
- `CohesionRadiusKey` - Float blackboard key receiving the cohesion radius (max ally distance from the centroid, cm).

24.1.10 `LcmBTService_ThreatFieldSampler`

Service that each tick samples the SVO SweepPenalty risk field at the agent's location, writing the threat level and an "in threat zone" flag (level >= AlertThreshold) into blackboard keys.

Properties

- `ThreatLevelKey` - Int blackboard key receiving the current SweepPenalty (0.255).
- `InThreatZoneKey` - Bool blackboard key receiving `Level >= AlertThreshold`.
- `AlertThreshold` - SweepPenalty value (0.255) at or above which the agent counts as in a threat zone.

24.1.11 `LcmBTTask_ChunkPrewarm`

Task that proactively forces SVO chunk generation at a target location and waits (up to TimeoutSeconds) for the chunk to become ready before succeeding.

Properties

- `TargetKey` - Blackboard key for the location whose chunk to prewarm. Vector = world location, Object = AActor (its current location is used).
- `bRequireFullResolution` - When true, the task waits for the chunk to be fully resolved (post-promotion); when false, any loaded chunk satisfies.
- `TimeoutSeconds` - Maximum time (seconds) to wait for the chunk before the task fails.

24.1.12 `LcmBTTask_DynamicObstaclePush`

Task that steers the agent down the SweepPenalty gradient (away from rising threat) until it clears the obstacle or times out.

Properties

- `PushSpeed` - Movement speed (cm/s) while pushing away from the obstacle.
- `ProbeStep` - Step distance (cm) used to sample the SweepPenalty gradient around the agent.
- `ExitThreshold` - SweepPenalty value (0.255) at or below which the agent is considered clear and the task succeeds.
- `MaxDuration` - Maximum time (seconds) to push before the task succeeds and yields.

24.1.13 `LcmBTTask_EvadeCBS`

Task that retreats from the threat gradient using context-based steering (CBS) over SVO danger sampling, blending an anti-threat direction with ray-evaluated safe headings until clear or timed out.

Properties

- `EvadeSpeed` - Movement speed (cm/s) while evading.
- `ProbeStep` - Step distance (cm) used to sample the SweepPenalty gradient around the agent.
- `ExitThreshold` - SweepPenalty value (0.255) at or below which the agent is considered safe and the task succeeds.
- `MaxDuration` - Maximum time (seconds) to evade before the task succeeds and yields.
- `AgentRadius` - Agent collision radius (cm) used when evaluating SVO danger for the CBS rays.
- `LookAheadDistance` - Distance (cm) ahead along each CBS ray over which SVO danger is sampled.
- `NumCBSRays` - Number of context-based-steering rays cast around the agent to select a safe heading.

24.1.14 `LcmBTTask_FindCover`

Behavior Tree task that locates a navigable SVO leaf with line-of-sight blocked from the blackboard enemy actor (i.e. cover). Returns Succeeded once a valid cover location is written back to the blackboard, or Failed if no cover within `SearchRadius` qualifies.

Properties

- `EnemyKey` - The Blackboard Key for the Enemy actor (Who are we hiding from?)
- `SearchRadius` - Search Settings (Exposed to Behavior Tree)
- `CoverHeightOffset` - Vertical offset (cm) added to candidate cover positions so the pawn stands above the cover floor rather than embedded in it.
- `bPrioritizeClosest` - When true, the closest qualifying cover position wins. When false, the search picks the first qualifying candidate (lower latency).

24.1.15 `LcmBTTask_FlyTo`

Behavior Tree task that flies an AI pawn from its current position to a blackboard target via the LCMNav3D pathfinder. Owns one async path request token (re-issued on dynamic replanning), runs per-tick steering (configurable flight style + avoidance style + wall avoidance + swarm separation), and finishes the task on arrival within `AcceptanceRadius`.

Functions

- `OnPathFound` - The Callback function for Async Pathfinding

Properties

- `AcceptanceRadius` - Settings exposed to the Behavior Tree Node
- `FlightSpeed` - Cruise speed of the pawn along the path, in cm/s.
- `QueryOptions` - Per-request pathfinding options (solver choice, smoothing, heuristic mode). Forwarded as-is into `FindPathAsync`.
- `SeparationRadius` - How close is "too close" to another agent? (e.g. 150 units)
- `SeparationWeight` - How hard should we push away from neighbors? (0.0 = No push, 2.0 = Strong push)
- `bUseFlightMovementComponent` - [P5] Delegate locomotion to a ULcmFlightMovementComponent on the pawn (for ultra-realistic 6-DOF drone/bird/fish dynamics). When ON and the pawn has an enabled component, this task only produces the guidance velocity (pure-pursuit + avoidance + arrival slowdown) and the component owns position + orientation. OFF (default) = the built-in kinematic integrator below. The kinematic-only knobs below (accel/banking/flight style) hide when this is ON - the component owns them via its FlightConfig.
- `MaxAcceleration` - How fast can we accelerate? (Higher = Snappier, Lower = Drifty/Heavy) [Kinematic only - owned by the flight component when delegating.]
- `BankingAmount` - How much do we bank into turns? (Visual only) [Kinematic only.]
- `MaxBankingAngle` - Max banking angle in degrees (e.g., 45 degrees) [Kinematic only.]
- `DecelerationDistance` - Distance to start slowing down (Arrival behavior). Still used when delegating - it shapes the guidance target speed near the goal.
- `FlightStyle` - Choose how this agent moves [Kinematic only - the component's FlightConfig has its own model/style when delegating.]
- `AvoidanceTimeHorizon` - How many seconds into the future do we look? (Standard: 2.0s)
- `CollisionMargin` - Radius multiplier for safety (e.g. 1.2 = Avoid by 20% extra margin)
- `AvoidanceStyle` - Choose how this agent avoids neighbors
- `WallCheckDistance` - WALL AVOIDANCE How far to check for walls? (e.g. 150 units)
- `WallPushForce` - How hard to push away from walls? (Must be strong!)
- `PathCenteringForce` - PATH CORRECTION How hard to pull agents back to the center line? (e.g. 500) 0 = Free drift (current behavior) 1000 = Strict line following (Train on tracks)
- `bEnableGoalPrediction` - When true, AND r.LcmNav.Chase.Enable != 0, the task resolves moving Object-key goals through FLcmGoalProvider and uses constant-velocity lead prediction (linear-extrapolation intercept) as the planner goal. When false, the legacy snapshot behavior is preserved bit-identically.

- `LeadSecondsCap` - Upper bound on lead time (seconds). Forwarded as LeadSecondsCap to FLcmGoalProvider::GetLeadPosition. Clamps blow-up when the target is receding faster than the agent can close. Override at runtime via r.LcmNav.Chase.LeadSecondsCap (>= 0). Typical 0.5 to 2.0.
- `DriftReplanCm` - Distance (cm) between the last-planned goal and the current lead position that triggers an automatic replan. Override at runtime via r.LcmNav.Chase.DriftReplanCm (>= 0).
- `bForceEnableChase` - [Fab marketplace UX, 2026-05-29] Per-task override for the `r.LcmNav.Chase.Enable` CVar gate. When true, chase activates from `bEnableGoalPrediction` alone - buyers don't need to know the CVar exists. Default false preserves the legacy two-gate behavior (CVar AND bEnableGoalPrediction) bit-identically. The CVar is retained as a project-wide kill switch / debug toggle.

24.1.16 `LcmBTTask_FollowFlowField`

Task that computes an LCM Nav3D path to the goal and follows its waypoints (LinearDirect steering) until arrival or timeout.

Properties

- `GoalKey` - Blackboard key for the goal. Vector = world location, Object = AActor (its current location is used).
- `MoveSpeed` - Movement speed (cm/s) along the followed path.
- `WaypointAdvanceRadius` - Distance (cm) within which the current waypoint is considered reached and the next is selected.
- `ArrivalRadius` - Distance (cm) from the goal at which the task succeeds.
- `AgentRadius` - Agent collision radius (cm) used by the path probe's clearance gate.
- `MaxDuration` - Maximum time (seconds) to follow the path before the task fails.

24.1.17 `LcmBTTask_Land`

Task that descends the agent vertically at DescentRate until it reaches the TouchDownZ world height, then succeeds.

Properties

- `TouchDownZ` - Target world Z height (cm) to descend to.
- `DescentRate` - Vertical descent speed in cm/s.
- `TouchDownTolerance` - Distance (cm) above TouchDownZ at which the agent counts as landed.

24.1.18 `LcmBTTask_OrbitTarget`

Task that circles the agent around a target, holding OrbitRadius while moving tangentially at OrbitSpeed.

Properties

- `TargetKey` - Blackboard key for the orbit centre. Vector = world location, Object = AActor (its current location is used).
- `OrbitRadius` - Distance (cm) the agent maintains from the target while orbiting.
- `OrbitSpeed` - Tangential movement speed (cm/s) around the orbit.
- `bClockwise` - When true, orbit clockwise (viewed from above); when false, counter-clockwise.
- `bMatchTargetZ` - When true, the agent matches the target's world Z height while orbiting; when false, it holds its own height.

24.1.19 `LcmBTTask_Patrol`

Task that moves the agent through a list of waypoints in order, pausing at each, optionally looping.

Properties

- `Waypoints` - Ordered list of patrol waypoints, in world coordinates.
- `MoveSpeed` - Movement speed (cm/s) between waypoints.
- `ArrivalRadius` - Distance (cm) within which a waypoint counts as reached.
- `DwellSeconds` - Time (seconds) the agent dwells at each waypoint before advancing.
- `bCycle` - When true, the patrol loops back to the first waypoint after the last; when false, it ends.

24.1.20 `LcmDynamicObstacleComponent`

Marks an actor as a dynamic LCMNav3D obstacle.

Two operating modes, selected by `r.LcmNav.DynObs.EventDriven`:

1. EVENT-DRIVEN (default, EventDriven=1) -- binds to `RootComponent->TransformUpdated`. Per-event the component unions the new bounding box into a pending dirty region and arms a one-shot coalesce timer (`r.LcmNav.DynObs.CoalesceMs`, default 33 ms = one frame at 30 fps). On timer expiry the union of old-footprint + all coalesced new-footprints is flushed in a single `UpdateDynamicObstacle` call. Component tick is disabled in this mode. Zero CPU cost when the actor is stationary; instant response (within one coalesce window) when it moves.
2. POLLING (EventDriven=0) -- legacy behavior: TickComponent at 10 Hz polls `GetActorLocation` and fires only when displacement exceeds `UpdateDistanceThreshold` (default 50 cm). Provided for ablation / deterministic-test scenarios where event jitter would perturb traces.

§P9.6 in

Functions

- `RemoveTrackedComponent` - Runtime/BP removal of a previously added sub-component. Components listed in TrackedComponents (the designer list) are not affected.
- `AddTrackedComponent` - Runtime/BP registration of an obstacle sub-component (e.g. a child mesh attached after BeginPlay). Idempotent; rebinds immediately.

Properties

- `UpdateDistanceThreshold` - [Polling mode only] How far does the object need to move before triggering an SVO rebuild? Keep around half your MinVoxelSize (~50 cm). Ignored when r.LcmNav.DynObs.EventDriven=1 (event-driven mode uses a coalesce timer instead of a distance threshold).
- `TrackedComponents` - Designer-facing list - pick the collision/mesh components that actually represent this obstacle. Empty = whole actor (legacy).

24.1.21 `EnvQueryContext_LcmKnownThreats`

EQS context exposing tag-marked threat actors as query targets for tactical tests.

Properties

- `ThreatTag` - Actors carrying this tag are treated as threats.

24.1.22 `EnvQueryContext_LcmThreatsFromAIPerception`

EQS context exposing the querier's AI-perception-tracked threats as query targets.

24.1.23 `EnvQueryGenerator_LcmHemisphericalRing3D`

Hemispherical-ring shell of unblocked 3D points around a context.

Properties

- `GenerateAround` - Context the shell is centred on (the target; defaults to the querier).
- `RadiusMin` - Inner radius of the sampling shell (cm).
- `RadiusMax` - Outer radius of the sampling shell (cm).
- `NumRadialSteps` - Number of concentric radial steps between RadiusMin and RadiusMax.
- `NumInclinationRings` - Number of inclination (polar/elevation) rings from horizon to pole.
- `NumAzimuthSectors` - Number of azimuth (longitudinal) sectors around each ring.
- `bUpperHemisphereOnly` - Restrict to the upper hemisphere (high-ground / perch queries).

24.1.24 `EnvQueryGenerator_LcmPointsAlongPath`

Unblocked points sampled along the LCM Nav3D path from one context to another.

Properties

- `FromContext` - Path start (defaults to the querier).
- `ToContext` - Path goal (set this to your destination context).
- `AgentRadius` - Agent capsule radius used to plan the LCM Nav3D path that is sampled (cm).
- `Spacing` - Distance between consecutive sample points along the path (cm).

24.1.25 `EnvQueryGenerator_LcmPortalProximity`

Candidate points at macro-graph portal centres near a context (chokepoints).

Properties

- `GenerateAround` - Context the portal search is centred on (defaults to the querier).
- `Radius` - Only portals within this radius of the context are emitted (cm).

24.1.26 `EnvQueryGenerator_LcmPointsInSVOVolume`

Generates uniform-density candidate points in an AABB, filtered to unblocked SVO volume leaves.

Properties

- `GenerateAround` - Context the sampling volume is centred on (defaults to the querier).
- `HalfExtent` - Half-extent of the sampling AABB (cm).
- `Spacing` - Grid spacing between candidate points (cm).

24.1.27 `EnvQueryTest_LcmLineOfSight3D`

Visible to the context via true-3D SVO line of sight. Boolean.

Properties

- `Context` - Context to test visibility against (defaults to known threats).
- `AgentRadius` - Agent capsule radius used for the volumetric SVO LOS sweep (cm).

24.1.28 `EnvQueryTest_LcmReachability3D`

An LCM Nav3D path exists from the context to the item. Boolean.

Properties

- `Context` - Context the path is planned from (defaults to the querier).
- `AgentRadius` - Agent capsule radius used when testing for a valid LCM Nav3D path (cm).

24.1.29 `EnvQueryTest_LcmPathDistance3D`

LCM Nav3D path distance (NOT euclidean) from the context to the item. Float (cm).

Properties

- `Context` - Context the path distance is measured from (defaults to the querier).
- `AgentRadius` - Agent capsule radius used to plan the path that is measured (cm).

24.1.30 `EnvQueryTest_LcmThreatExposure`

Summed §P9.3 SweepPenalty along the item→threat segments. Float (lower = safer).

Properties

- `Context` - Threat context whose members the exposure is summed against (defaults to known threats).
- `NumSamplesPerSegment` - Number of sample points evaluated along each item->threat segment.

24.1.31 `EnvQueryTest_LcmCoverScore3D`

Fraction of context threats the item is occluded from. Float (higher = better cover).

Properties

- `Context` - Threat context the item must be occluded from to count as cover (defaults to known threats).
- `AgentRadius` - Agent capsule radius used for the occlusion LOS sweep to each threat (cm).

24.1.32 `EnvQueryTest_LcmHeightAdvantage`

Vertical advantage of the item over the context (high ground). Float.

Properties

- `Context` - Context the item's vertical advantage is measured against (defaults to known threats).

24.1.33 `LcmFlightMovementComponent`

Opt-in pawn movement component that turns nav guidance velocities (RequestVelocity from FlyTo tasks) into realistic 6-DOF flight/swim motion via ULcmFlightDynamics, sweeps the transform, and broadcasts the anim state. Inert until bEnabled is set.

Functions

- `RequestVelocity` - Feed the guidance command for this frame (world-space desired velocity). Persists until replaced or cleared, so a slower controller tick still produces continuous motion.
- `GetRigidState` - The full 6-DOF integrator state from the last step.
- `GetFlightState` - The animation / secondary-motion drive signals from the last step.
- `ClearGuidance` - Stop consuming guidance and coast (dynamics still decelerate the body).

Properties

- `bEnabled` - Master opt-in. Default false → the component is inert (base behaviour), guaranteeing byte-identical motion to before it was added.
- `FlightConfig` - Body/aero/control tuning. The LocomotionModel + Archetype selectors live at the top of this struct and drive which parameters are shown (only the selected model/archetype's fields appear).
- `bBroadcastAnimState` - [P6] Each tick, push the flight state to the pawn's AnimInstance (and the owner) if it implements ILcmFlightAnimInterface. Default on; costs nothing unless an implementer exists.
- `RigidState` - Persistent integrator state, seeded from the pawn at BeginPlay and re-synced from the updated component each tick (so external teleports and swept-collision corrections are respected).
- `FlightState` - Last-step anim outputs.
- `PendingDesiredVelocity` - Latest guidance command and whether one is active.
- `bHasGuidance` - True when RequestVelocity was called since the last tick - gates the dynamics step so an unfed component coasts instead of integrating stale guidance.

24.1.34 `LcmFlyingPawn`

Note: APawn already inherits from INavAgentInterface - re-listing it here triggers C4584 (duplicate base class). We just override the virtuals.

Properties

- `CollisionSphere` - Root sphere collision component for the flying pawn.
- `MovementComponent` - LCM Nav3D-aware floating movement component driving the pawn along nav paths.

24.1.35 `LcmGameplayAbility_RequestPath`

GameplayAbility wrapper around the LCM Nav3D async path request, gating it through the standard GAS cost/cooldown/cancellation pipeline (budget via ULcmNavAttributeSet) and broadcasting the resulting path on completion.

Functions

- `OnApexPathFound` - Async FindPathAsync callback (game thread): populates the result, broadcasts OnPathReady, and ends the ability.

Properties

- `StartLocation` - Path-search start.
- `GoalLocation` - Path-search goal.
- `QueryOptions` - Forwarded to FindPathAsync verbatim.
- `BudgetCost` - Cost in PathRequestBudget consumed per activation. Default 1. Reduce for "cheap" replans (e.g. drift-triggered) and raise for expensive cross-chunk queries.
- `OnPathReady` - Fired on completion (success or failure). Subscribers should receive PathPoints by-const-ref.

24.1.36 `LcmMassCrowdSpawner`

World-Partition-safe Mass spawner: never streams out, so its crowd's lifetime is not tied to the player's distance from the spawn point (fixes the "all agents disappear when the player leaves the area" WP trap).

Properties

- `SpawnBatchPerFrame` - [startup smoothing, opt-in] Spawn at most N entities per frame instead of the whole crowd in one frame. 0 (default) = stock engine behaviour. A large one-frame spawn (e.g. 6000 agents) hitches startup ~70 ms; spreading it (e.g. 500) smooths that frame-pacing spike. Only applies when "Auto Spawn On Begin Play" is enabled. NOTE: this does NOT fix the editor's "Handled ensure: CurrentPhase==." Mass phase-manager spam - that is an engine-level startup race (observed on 5.2) (fires with 2 agents, on finite maps, independent of spawn load; handled + compiled out of Shipping), not a spawn-frame cost.

24.1.37 `LcmMassFlowSteerTrait`

[DEPRECATED - M3, 2026-07-03] Legacy flow-field crowd trait. Superseded by the unified `ULcmMassNavAgentTrait` ("LCM Nav3D Agent") with NavMode=FlowField, which composes a byte-identical archetype. Kept fully functional; prefer the unified trait for new content. Authoring trait for a flow-field-steered crowd agent.

Properties

- `GoalMode` - FixedLocation (use Goal) or TaggedActor (chase an actor by tag - the crowd follows it as it moves; the shared field re-solves on the move).
- `Goal` - Shared crowd goal (world space, FixedLocation mode). All agents with this goal share one field.
- `TargetActorTag` - Tag of the moving target actor (TaggedActor mode).
- `bChaseLead` - Lead the moving target by its velocity (intercept ahead of it).
- `LeadSecondsCap` - Upper bound on lead time (s).
- `MaxSpeed` - Movement speed (cm/s).
- `AgentRadius` - Collision radius (cm) - also the minimum agent size for field generation.
- `AvoidanceRadius` - Neighbours within this distance contribute local avoidance steering (cm).
- `AvoidanceStrength` - Weight of avoidance vs flow-following (0 = off, 1 = balanced).
- `MaxAcceleration` - Max acceleration (cm/s²). Smooth ease in/out; 0 = instant (legacy floaty). Applies only to PhysicsWeighted flight (LinearDirect is instant).
- `MaxTurnRateDeg` - Max turn rate (deg/s). The key knob: smooth banking turns vs robotic snapping. Applies only to PhysicsWeighted flight (LinearDirect is instant).
- `ArriveRadius` - Arrival / slow-down radius around the goal (cm).
- `FlightStyle` - LinearDirect (instant) vs PhysicsWeighted (turn-rate + acceleration inertia).
- `AvoidanceStyle` - None / Reactive (boids) / Predictive (ORCA, smooth mutual passing) / CBS.
- `AvoidanceTimeHorizon` - ORCA time horizon (s) - Predictive avoidance only.
- `LaneBias` - Lateral lane-spread strength (0 = off). r.LcmNav.Mass.FlowLaneBias (>=0) overrides.
- `LaneDensityNorm` - Neighbour count at which lane spreading saturates.
- `bAnticipatory` - ANTICIPATORY navigation: route the crowd around where moving obstacles are HEADING (their predicted swept path), not just where they are now - so it goes OVER a side gap that is about to close instead of squeezing into it. Costs a little more (predicted-occupancy stamping + slightly more re-solving) and can detour a touch wider. Off = today's reactive field. (r.LcmNav.Mass.Anticipatory overrides this: 0 forces off, >=1 forces on.)
- `AnticipatoryStrength` - How wide a berth to give predicted obstacle occupancy (higher = earlier / wider detour).

24.1.38 LcmMassGoalTrait

Authoring trait: assign a fixed-location or tagged-actor goal to a Nav3D Agent so it auto-paths there (requires the "LCM Nav3D Agent" trait too).

Properties

- `GoalMode` - FixedLocation (use GoalLocation) or TaggedActor (path to an actor by tag).
- `GoalLocation` - World location goal (FixedLocation mode).
- `TargetActorTag` - Tag of the target actor (TaggedActor mode). First actor with this tag wins.
- `RepathToleranceCm` - Re-issue the path if the resolved goal drifts more than this (cm).
- `ArrivalToleranceCm` - Treat the agent as "arrived" within this distance; stop re-pathing (cm).
- `ReissueIntervalSeconds` - Minimum seconds between path re-issues per agent (anti-flood safety).
- `bEnableGoalPrediction` - [Chase] Lead a moving TaggedActor target by its velocity so agents intercept rather than trail. TaggedActor mode only. Master kill switch: r.LcmNav.Mass.Chase.Enable.
- `LeadSecondsCap` - [Chase] Upper bound on lead time (s). Override fleet-wide with r.LcmNav.Mass.Chase.LeadSeconds (>=0).

24.1.39 LcmMassNavISMRenderTrait

Authoring trait: drop on the Mass config, set a mesh, get instanced visuals.

Properties

- `Mesh` - Mesh drawn (instanced) at each agent's transform.
- `UniformScale` - Uniform scale applied to each instance.

24.1.40 LcmMassNav3DTrait

[DEPRECATED - M3, 2026-07-03] Legacy path-based agent trait. Superseded by the unified `ULcmMassNavAgentTrait` ("LCM Nav3D Agent") with NavMode=PathFollowing, which composes a byte-identical archetype. Kept fully functional so existing configs (DA_MassAgent_Apex) keep working; prefer the unified trait for new content.

Authoring trait: attaches the path nav fragments and exposes radius/speed, pathfinding, and natural-movement settings.

Properties

- `AgentRadius` - Collision radius forwarded to the solver (cm).
- `MaxSpeed` - Maximum movement speed for the steering processor (cm/s).
- `MobilityClassIndex` - Mobility-class index (reserved for per-class cost rules; 0 = default).
- `AvoidanceRadius` - Neighbours within this distance contribute local avoidance steering (cm).
- `AvoidanceStrength` - Weight of avoidance vs path-following (0 = off, 1 = balanced).
- `PathSolver` - Any-angle LazyThetaStar (smooth) vs grid-locked AStar (square turns).
- `SmoothingMode` - Curved (CatmullRom) / shortcut (Linear) / raw staircase (None).
- `MaxAcceleration` - Max acceleration (cm/s^2). Limits speed change so agents ease into motion.
- `MaxTurnRateDeg` - Max turn rate (deg/s). The key knob: replaces robotic square turns with smooth arcs. Lower = wider, lazier turns; 0 = instant snapping (legacy).
- `LookAheadDistance` - Pure-pursuit look-ahead distance (cm). Larger = smoother cornering.
- `ArriveRadius` - Arrival / slow-down radius around the goal (cm).
- `FlightStyle` - LinearDirect (magic/biological - instant) vs PhysicsWeighted (inertia).
- `AvoidanceStyle` - None / Reactive (boids) / Predictive (ORCA, smooth passing) / CBS.
- `AvoidanceTimeHorizon` - ORCA time horizon (s) - used only by Predictive avoidance.
- `LaneBias` - Density-aware lateral lane-spread strength (0 = off).
- `LaneDensityNorm` - Neighbour count at which lane spreading saturates.
- `bObstacleReactivity` - Re-plan when a moving obstacle invalidates the route this agent is following. Global kill switch: r.LcmNav.Mass.ObstacleReactivity.

24.1.41 `LcmMassNavAgentTrait`

[M3] Unified authoring trait: one trait, a NavMode selector. PathFollowing composes the path archetype (NavAgent + PathRequest + PathResult); FlowField composes the flow archetype (FlowField state + a const shared config fragment). Mode-specific properties hide via EditCondition on NavMode.

Properties

- `NavMode` - Movement policy. PathFollowing = per-agent A* path; FlowField = shared crowd steering field. Drives which fragments + config + tag the trait adds.
- `Compute` - Compute selector governing both modes. This release solves on the CPU, so `Auto` and `CPU` behave identically; the selector is retained for forward compatibility.
- `AgentRadius` - Collision radius (cm).
- `MaxSpeed` - Maximum movement speed (cm/s).
- `AvoidanceRadius` - Neighbours within this distance contribute local avoidance steering (cm).
- `AvoidanceStrength` - Weight of avoidance vs path/flow following (0 = off, 1 = balanced).
- `MaxAcceleration` - Max acceleration (cm/s^2). 0 = instant.
- `MaxTurnRateDeg` - Max turn rate (deg/s) - replaces robotic square turns with smooth arcs. 0 = instant.
- `ArriveRadius` - Arrival / slow-down radius around the goal (cm). NOTE: the legacy traits defaulted differently per mode (path 80 / flow 150); this unified default is 100 - set explicitly if migrating a config that relied on the old default.
- `FlightStyle` - LinearDirect (instant, no inertia) vs PhysicsWeighted (turn-rate + accel inertia).
- `AvoidanceStyle` - None / Reactive (boids) / Predictive (ORCA) / CBS. NOTE: the legacy Flow Steer trait defaulted to Predictive (ORCA); this unified default is Reactive (matching the legacy path trait) - pick Predictive explicitly for smooth mutual passing in dense flow crowds.
- `AvoidanceTimeHorizon` - ORCA look-ahead horizon (s) - Predictive avoidance only.
- `LaneBias` - Density-aware lateral lane-spread strength (0 = off).
- `LaneDensityNorm` - Neighbour count at which lane spreading saturates.
- `MobilityClassIndex` - Mobility-class index (reserved for per-class cost rules; 0 = default).
- `PathSolver` - Any-angle LazyThetaStar (smooth) vs grid-locked AStar (square turns).
- `SmoothingMode` - Curved (CatmullRom) / shortcut (Linear) / raw staircase (None).
- `LookAheadDistance` - Pure-pursuit look-ahead distance (cm). Larger = smoother cornering.
- `bObstacleReactivity` - Re-plan when a moving obstacle invalidates the route being followed.
- `FlightConfig` - Body / aero / control tuning for the 6-DOF model - the SAME model as the pawn Lcm Flight Movement Component. The LocomotionModel + Archetype selectors are at the top of this struct and drive which parameters show; default LocomotionModel = Kinematic → the crowd moves exactly as before until you opt in.
- `FlightDynamicsLODDistance` - [P8] Dynamics LOD distance (cm): beyond this from the viewer, agents use the cheap kinematic integrator (full 6-DOF only near camera) - the scale knob for large crowds. 0 = always full. Kill switch: r.LcmNav.Mass.FlightLOD.
- `bAnticipatory` - Route around where moving obstacles are HEADING (predicted swept volume).
- `AnticipatoryStrength` - Detour strength around predicted occupancy (higher = wider berth).
- `bAvoidThreats` - Route the crowd AROUND actors tagged "ApexThreat" (danger zones). NOTE: agents DETOUR around threats, which can slow arrival - measured -30% threat crossings but 13-25% fewer arrivals on tight corridors. Enable for threat/danger gameplay.
- `ThreatDetourStrength` - Threat detour strength (higher = wider berth around danger zones).
- `bIncrementalField` - [Perf] D* Lite INCREMENTAL corridor field for large / churn-heavy crowds (re-cost only changed cells; safe full-rebuild fallback). Shared per goal.
- `bAsyncFlowSolve` - [Perf P2] Solve the flow field on a WORKER thread (removes solve spikes from the frame; the field publishes a frame later). Recommended for large crowds.

24.1.42 `LcmMassPerceptionTrait`

Authoring trait: makes an LCM Nav3D Mass agent both a senser and a perceivable (faction-based SVO-LOS perception), exposing sight/FOV/eye-height settings.

Properties

- `FactionId` - Faction id (different factions are mutually hostile / perceivable).
- `SightRadius` - Maximum sight distance (cm).
- `FOVDegrees` - Full field-of-view angle (degrees).

- `EyeHeightOffset` - Eye height offset from the agent origin (cm).
- `AgentRadius` - Agent radius for the SVO LOS walk (cm).
- `PerceptionIntervalFrames` - Perceive once every N frames (LOD throttle).

24.1.43 `LcmMassRepresentationTrait`

Preconfigured UMassVisualizationTrait: LOD-switched crowd rendering that draws the instanced static mesh at all visible LODs by default (no "see nothing" trap), and self-provides the LOD-collector + actor fragments so it works as one trait.

24.1.44 `LcmNavAreaModifier`

Designer-placed 3D volumetric cost-overlay actor; applies a cost multiplier to LCM Nav3D pathfinding wherever the agent's leaf is inside its box (water, storm, danger, or preferred-lane volumes that stack multiplicatively).

Functions

- `GetCostMultiplier` - Returns this volume's cost multiplier applied to agents inside the box.
- `GetBounds` - World-space FBox covering the modifier volume.

Properties

- `BoundsBox` - Box bounds - extents around the actor's location. Use the editor gizmo to size the volume.
- `CostMultiplier` - Cost multiplier applied to LCM Nav3D pathfinding when the agent is inside the box. 1.0 = no effect; 2.0 = costs twice as much; 0.5 = preferred lane. Default 2.0.
- `bAutoRegisterWithApex` - Auto-register with the LCM Nav3D extension registry on BeginPlay.
- `AreaTag` - Optional gameplay tag for designer categorisation (e.g. "Water", "Storm", "Danger" - not used by solver in v1).

24.1.45 `LcmNavAreaQueryLibrary`

Blueprint function library exposing the LCM Nav3D nav-extension registry so designers can query cost overlays and registered NavLinks from BP graphs.

Functions

- `GetRegisteredLinkCount` - Count of currently-registered NavLinks.
- `GetRegisteredAreaCount` - Count of currently-registered NavArea modifiers.
- `GetCostMultiplierAt` - Cost multiplier at the given world position. Returns 1.0 when no modifier applies; multiplies when overlapping modifiers stack.
- `GetCostMultiplierAlongPath` - Returns the sum of per-leg cost multipliers along the waypoint chain, weighted by leg length. Useful for "is this path expensive" predicates without rerunning the solver.
- `GetAllLinkEndpoints` - All registered link endpoints as FVector pairs (Start, End). Useful for debug visualisation.
- `FindNearestNavLink` - Finds the nearest registered NavLink to `WorldPos` within `MaxDistanceCm`. Returns nullptr if none.

24.1.46 `LcmNavArmTagLibrary`

Blueprint-callable wrapper around the `r.LcmAblation.ArmTag` CVar so designers can stamp per-task arm tags on telemetry rows for in-game A/B analysis without using the console.

Functions

- `SetArmTag` - Sets the current `r.LcmAblation.ArmTag` value. Empty FName clears it.
- `PushArmTag` - Atomically swap the arm tag to `NewArmTag`, returning the previous value. Caller pairs with PopArmTag to restore on task exit.
- `PopArmTag` - Restore a previously-pushed arm tag.
- `GetCurrentArmTag` - Reads the current `r.LcmAblation.ArmTag` value. Empty FName when unset.

24.1.47 LcmNavAttributeSet

GAS attribute set surfacing per-agent path-request budgets so projects can throttle expensive LCM Nav3D queries through the standard GameplayEffect pipeline.

Functions

- `OnRep_PathRequestBudget` - RepNotify for PathRequestBudget; mirrors the replicated change to GAS.
- `OnRep_MaxPathRequestBudget` - RepNotify for MaxPathRequestBudget; mirrors the replicated change to GAS.

Properties

- `PathRequestBudget` - Current path-request budget. Consumed by 1 per successful ULcmGameplayAbility_RequestPath activation. Default 5.
- `MaxPathRequestBudget` - Hard cap clamp for regen. Default 5.

24.1.48 LcmNavigationInvokerComponent

Attach this to the Player or VIPs. The SVO Manager will only generate voxel data for chunks within this radius.

Properties

- `GenerationRadius` - How far around this actor should we build navigation data? E.g., 10000 = Build chunks within a 100m radius.

24.1.49 LcmNavigationManagerSVO

LCMNav3D world singleton - the central navigation manager actor. Spawns the four subsystem components (`ULcmSVOStreamingComponent`, `ULcmSVOGenerationComponent`, `ULcmPathfindingComponent`, `ULcmSVORenderComponent`); owns the SVO data containers (`FiniteData` for finite worlds, `ActiveVirtualChunks` for infinite worlds); owns `FLcmMacroGraph` + `FLcmPortalBuilder`; hosts the multicast RPCs for dynamic-obstacle invalidation and flow-field-ready notifications. Exactly one instance per UWorld; auto-discovered via `UGameplayStatics:: GetActorOfClass`. Full responsibility list in §P2.2 of `Docs/SYSTEM_ARCHITECTURE.md`.

Functions

- `UpdateDynamicObstacleReplicated` ---------- --- Server-authoritative entry: applies the dynamic-obstacle update locally AND multicasts it to all connected peers (clients + listen-server). Safe to call from any context -- becomes a no-op + warning when called on a client (clients route writes through the server). The flow-field-ready signal is a multicast-only callback path. The flow-field buffers are NOT replicated -- this RPC informs peers that a goal at (GoalLocation, ChunkKey) is reachable on the server, so they can prewarm their own local generation or trigger gameplay.
- `UpdateDynamicObstacle` - Call this when a large object moves to update the nav graph locally.
- `ToggleDrawDebugVolumes` - Flips bDrawDebugVolumes via SetDrawDebugVolumes. Convenience entry for a single-key toggle binding.
- `SetMacroTopologyCell` - [Procedural Topology] Called by your procedural-generation system (or World Partition minimap scanner) to mark a coarse cell blocked/unblocked.
- `SetDrawDebugVolumes` - Runtime show/hide for the SVO debug voxel volumes. Sets bDrawDebugVolumes and ALWAYS rebuilds the render proxy. The scene proxy captures the flag at build time (FLcmSVOSceneProxy::bForceDraw), so a rebuild is required for BOTH show and hide - toggling the bool alone does nothing. Bind to a key for in-game/PIE show/hide.
- `Multicast_UpdateDynamicObstacle` - Server-only entry: multicasts a dynamic-obstacle update to every connected client so each peer's `ALcmNavigationManagerSVO` replays `UpdateDynamicObstacle(Bounds)` locally (RCU clone is identical on every host). Clients that fire this directly are warn-and-ignored.
- `Multicast_FlowFieldReady` - Server-only entry: multicasts a "flow-field finished generating" notification to clients so they can prewarm a local sample for the given goal + chunk. The flow-field buffers are NOT replicated; clients regenerate on demand using shared cache keys.
- `ForceRedrawSVO` - Keep the core function so Blueprints can still call it at runtime if needed
- `FindPath` - Calculates a path from Start to End using the SVO.
- `FindCover` - Find a hidden spot safe from the Enemy.

- `DrawDebugSVO` - Visualizes the octree in the viewport (Green = Free, Red = Blocked). One-shot manual call: flushes prior persistent lines and draws with a 10 s lifetime. The per-frame Tick path uses DrawDebugSVOTick instead.
- `DrawDebugNeighborsAt` - Visualizes neighbors of the node at the specific location (for debugging connectivity).
- `DrawDebugInfiniteWorld` - Renders the real-time state of the infinite world systems.
- `DebugDrawFlowField` - Flow Field Debug Visualization Generates a temporary Flow Field to the Target-Location and draws it as a 3D Vector Heatmap. Green = Low Cost (Near Target), Red = High Cost (Far from Target).
- `BuildSVOInEditor` - Editor-time SVO build (button in the Details panel). Populates FiniteData from live collision geometry WITHOUT entering PIE, so the stock UE EQS Testing Pawn - and any LCM Nav3D EQS query built from the generators & tests - resolves the SVO and renders true-3D candidate points in the editor viewport, exactly the in-editor workflow Recast navmesh gives for 2.5D EQS. Finite mode only; warns + no-ops while playing (the SVO already bakes on BeginPlay) and in infinite-world mode (which streams chunks at runtime). Press once, then drop an EQSTestingPawn. // Editor-time SVO build. Surfaced as the prominent "Build SVO" button in the // editor module's banner (FLcmNavManagerDetails); BlueprintCallable so it can // also be triggered from script. Intentionally NOT a CallInEditor button - that // rendered a detached row below the property sections instead of in place.
- `BuildSVO` - Clears old data and builds a new SVO from scratch (Finite Mode only).

Properties

- `bIsInfiniteWorld` - World Type. OFF = a single fixed-bounds SVO built for the whole level (best for arenas / linear levels). ON = a streamed, unbounded world using Hierarchical HPA* (chunks generate on-demand around invokers). Drives the Generation Strategy below.
- `GenerationMode` - Auto-derived from the World Type - NOT set by hand. An infinite world always uses the Hierarchical HPA* strategy (macro routing + on-demand chunk streaming), the only infinite mode; a finite world is always Static Finite. Shown read-only so it can never be set to a value that contradicts the World Type (e.g. infinite + Fixed Bounds, which would silently break streaming). Tick "Infinite World" above to switch strategy.
- `bAutoBuildOnPlay` - Should the SVO automatically build itself when the game starts? (Finite mode only)
- `WorldExtent` - The total size of the navigable world (Cube width). Only used for Finite worlds.
- `ChunkSize` - Size of each chunk in World Units (e.g., 50,000 for 500m chunks). Must match World Partition Grid.
- `MinVoxelSize` - The smallest possible voxel size. Recursion stops here. (e.g., 100 = 1 meter)
- `ClearancePadding` - The padding added to voxel collision checks. Set this to your largest agent's radius.
- `ObstacleQueryChannels` - Physics collision channels to treat as solid walls.
- `AllowedObstacleClasses` - If empty, ALL overlapping geometry is voxelized. If populated, ONLY these actor classes are voxelized.
- `CoarseCellSize` - The size of a single topological macro-cell. 2500 units (25 meters) is the AAA standard for coarse LOD pathfinding.
- `bUseFlowFields` - When true, the request classifier can elect the flow-field path instead of per-agent A* for swarms of ≥ `FlowFieldThreshold` agents sharing a goal. Off by default; opt-in for swarm gameplay.
- `FiniteObstacleUpdateInterval` - [Perf fix, 2026-06-07] FINITE mode: minimum seconds between dynamic- obstacle SVO applications. In finite mode every UpdateDynamicObstacle deep-clones the ENTIRE world SVO (RCU copy-on-write) - with event-driven obstacle components flushing every ~33 ms per moving actor, that was a clone storm ("massive fps drop while anything moves"). Updates arriving inside the window are UNIONED and applied as ONE clone+swap at the trailing edge, so the final state always lands. 0 = legacy immediate behaviour. Infinite mode is unaffected (per-chunk updates are cheap by construction).
- `FlowFieldThreshold` - Minimum number of agents sharing a goal before the classifier elects the flow-field path instead of per-agent A* (requires bUseFlowFields).
- `FlowFieldClusterCellSize` - Voxel-quantization grid size (cm) used to bucket flow-field goal positions into cache keys. Two requests whose goals fall into the same cell share a cached flow field. Larger values → more sharing (cheaper) but coarser direction snapping.

- `bUseGPUVoxelization` - Uses Compute Shaders for lightning-fast voxelization. Disable only if targeting low-end mobile.
- `MaxGenerationsPerFrame` - How many chunks can we voxelize per frame? (1 is usually best to avoid Game Thread stutters).
- `MaxFlowSolvesPerFrame` - [Perf P3a] Cap goal-anchored flow-field solves STARTED per frame (spreads the GT cost so a burst cannot spike the frame). 0 = unlimited. r.LcmNav.Mass.MaxCorridorSolvesPerFrame overrides (>=0).
- `MaxPathRequestsPerFrame` - [Perf] Max PathFollowing path requests dispatched per frame (spreads a spawn burst across frames; requests are async). r.LcmNav.Mass.MaxPathsPerTick overrides (>=0).
- `MaxRequestsPerFrame` - How many paths can we calculate per frame? (50 is a safe number to keep FPS high).
- `bChunkQuantizedMacroCache` - Share one cross-chunk macro route across a crowd heading to the same goal, instead of recomputing the identical chunk-level route per agent. This is the big win for LARGE Mass/AI crowds in infinite/hierarchical worlds: it removes the per-agent macro-pathfinding cost that otherwise spikes the game thread when many agents spawn or replan at once (measured ~30x cheaper request processing, worst-frame 97ms -> 12ms at 2000 agents). The macro route is chunk-level; each agent's exact start/end are still applied per agent, so paths stay correct. Leave OFF to reproduce legacy bit-exact pathfinding (e.g. for deterministic benchmarking). Runtime override: r.LcmNav.MacroCache.ChunkQuantized (force-on). Infinite/hierarchical only.
- `bDrawDebugVolumes` - If true, automatically renders the voxel grid when generation finishes.
- `bDrawInfiniteWorldDebug` - If true, draws Invoker Bubbles, the Generation Queue, and loaded chunks in real-time.
- `bDrawPathfindingDebug` - Master switch for Agent Flight Paths and Waypoints.
- `DebugRedrawCoalesceSeconds` - Minimum time between debug-mesh redraws (coalesce cadence). Forwarded to the SVO debug renderer; higher = fewer rebuilds when many obstacles move at once.
- `MacroGraph` - A lightweight map of every chunk in the world (HPA*).
- `RenderComponent` - The Central Renderer used to draw the SVO via the GPU Render Thread.
- `DynamicMacroTopology` - [Procedural Topology] Thread-safe map of the global layout of massive blocking geometry (coarse-grid coordinate -> blocked). Populated by SetMacroTopologyCell.
- `StreamingComponent` - SUBSYSTEM COMPONENTS
- `GenerationComponent` - SVO voxelization scheduler subsystem. Pulls pending chunk-generation requests off the manager's queue and dispatches CPU/GPU voxelization.
- `PathfindingComponent` - Pathfinding request queue + dispatcher subsystem. Routes async `FindPathAsync` calls to the CPU worker pool, per the request options.

24.1.50 `LcmNavLinkComponent`

3D vertical NavLink component (subclass of UNavLinkCustomComponent) with designer-set world-space LinkStart/LinkEnd for ladders, shafts, and teleporters that Recast's surface-projected links cannot represent.

Functions

- `IsVerticalLink` - True iff this link spans more than `MinVerticalCm` along Z - the 3D-advantage case (ladders, jump shafts).
- `GetLinkSpanCm` - Convenience: distance(LinkStart, LinkEnd) - useful for designers comparing the link's geometric length vs its travel cost.

Properties

- `LinkStart` - World-space start point. For ladders/teleporters this is the bottom (or originating end if not vertical).
- `LinkEnd` - World-space end point. For vertical ladders this is the top.
- `bBidirectional` - When true (default), the link can be traversed Start→End AND End→Start. Set false for one-way patterns (teleporter, drop-down, etc.).
- `bAutoRegisterWithApex` - Auto-register with the LCM Nav3D link registry on BeginPlay so v2 solver integration discovers the link without buyer code.
- `TraversalCostCm` - Traversal cost (cm-equivalent) added to the path through this link. Default 0 = no penalty (geometric distance counts as-is).

24.1.51 LcmNavVolume

Defines the playable area for offline SVO generation. Acts as the master bounds for slicing and baking ALcmNavigationChunk actors.

Functions

- `BakeNavigationData` - Editor-only one-shot bake. Slices the volume bounds into the configured chunk grid, voxelizes each chunk on the game thread, and writes a `ULcmSVODataAsset` per chunk into `OutputAssetPath`. Pairs with the runtime `ALcmSVOStreamingProxy` actor that loads each asset on level streaming.

Properties

- `ChunkSize` - Edge length (cm) of a single bake chunk. Must match the runtime manager's `ChunkSize` when `bSnapToGlobalGrid` is true.
- `MinVoxelSize` - Smallest voxel produced by SVO subdivision (cm). Tighter values produce finer paths but inflate the SVO node count cubically.
- `ClearancePadding` - Inflation (cm) applied to obstacle SAT collision checks at bake time. Set to the radius of your largest agent so the SVO carves out enough clearance for them.
- `bSnapToGlobalGrid` - If true, forces chunks to align to a global 3D grid (Required for World Partition). If false, spawns exactly ONE chunk that perfectly matches this volume's bounds (Best for Arena/Linear games).
- `BakeObstacleChannels` - Physics collision channels treated as solid obstacles at bake time. Empty means "all overlapping geometry".
- `BakeAllowedClasses` - Whitelist of actor classes to voxelize. If empty, all overlapping geometry on the configured channels is included.
- `OutputAssetPath` - Content path under which `ULcmSVODataAsset` outputs are written by `BakeNavigationData`. Per-chunk assets get a `_X_Y_Z` suffix.

24.1.52 LcmSmartObjectBehaviorDef_CoverPocket

SmartObject behaviour definition for a designer-placed cover pocket, validated at claim time by SweepPenalty + threat-hint LOS with a FindBestCover fallback when the placed pocket has gone hot.

Properties

- `DwellSeconds` - Time the pawn stays in cover before considering the state complete. 0 = stay forever (use a parent transition condition to exit).
- `MaxAcceptableSweepPenalty` - Highest SweepPenalty (0.255) the slot leaf may carry to count as safe. Above this the StateTree task triggers the FindBestCover fallback. Default 32 = "lightly swept, mostly safe".
- `bUseThreatHint` - Direction hint (world-space) to validate cover against. The pocket is only valid if LCM Nav3D LOS from this point to the slot is blocked AND SweepPenalty is below the cap.
- `ThreatHintLocation` - World-space point the cover is validated against; the pocket is valid only if LCM Nav3D LOS from here to the slot is blocked.
- `FallbackSearchRadiusCm` - When the placed pocket fails validation, the task searches within this radius for a procedural cover via FindBestCover. 0 disables fallback (failure makes the task return Failed).
- `AgentRadius` - Agent radius for the LOS walker's clearance gate.

24.1.53 LcmSmartObjectBehaviorDef_PatrolWaypoint

SmartObject behaviour definition for one patrol waypoint; designers chain placed waypoints via NextWaypointTag to author a patrol route as content.

Properties

- `DwellSeconds` - Dwell time at this waypoint before moving to the next.
- `NextWaypointTag` - GameplayTag of the next smart object in the chain. Empty = end-of- chain; the parent state machine drops out of patrol on Empty.
- `bChainCycles` - When true, the patrol restarts at the chain's head when the chain ends. When false, the pawn lingers at the final waypoint.

- `bUseFacingHint` - Optional facing yaw to assume at this waypoint (degrees). When unset (kept default), keeps the inbound flight rotation.
- `FacingYawDeg` - Yaw (degrees, world-space) the pawn faces at this waypoint when bUseFacingHint is set.

24.1.54 `LcmSmartObjectBehaviorDef_Perch`

SmartObject behaviour definition for a volumetric perch: fly here, hover or stand, and dwell - a 3D-only smart-object pattern LCM Nav3D's free-space SVO supports where surface-projected Recast navmesh cannot.

Properties

- `DwellSeconds` - How long the pawn dwells at the perch after arrival.
- `RequiredClearanceCm` - Agent collision radius the perch must accommodate. Used at claim time by `FLcmSTTask_UseSmartObject` to reject perches whose voxel clearance is below this threshold. Pass-through to the JFA clearance gate (same byte the Lazy Theta* LOS walker consumes).
- `bUseFacingHint` - When true, the pawn rotates to face `FacingHint` after arrival. When false, the pawn keeps its inbound flight rotation.
- `FacingYawDeg` - Yaw the perched pawn should face (degrees, world-space). Bound only when `bUseFacingHint` is true.
- `FacingInterpSpeed` - Per-tick speed at which the pawn aligns to FacingYawDeg during dwell.

24.1.55 `LcmSVODataAsset`

A lightweight, streamable asset that holds pre-calculated SVO navigation data. Designed to be loaded asynchronously by World Partition.

Properties

- `ChunkCoordinate` - The mathematical grid coordinate this chunk belongs to
- `ChunkBounds` - The physical world bounds of this chunk
- `SerializedSVOData` - The raw, compressed binary data of the SVO tree
- `SerializedMacroNode` ---------- Optional macro-graph slice for this chunk: the just-finalized FLcmMacroNode (PortalEdges + InternalEdges + WorldCenter) captured after BuildTruePortals + PrecomputeIntraChunkEdges complete during the bake. Serialized via FLcmMacroNodeArchive (magic 'LCMN', V1). Empty array = legacy bake - streaming proxy falls back to UpdateMacroNode + GeneratePortalsForChunk (fallback portals only; the runtime BuildTruePortals + intra-chunk-edge precompute still runs separately when neighbors load). Non-empty = the cold-build work is skipped and the precomputed macro node is injected directly into Manager->MacroGraph.Nodes[ChunkCoordinate]. Independent from SerializedSVOData - backward-compatible addition; older baked assets deserialize with this field empty.

24.1.56 `LcmTacticalQueryLibrary`

Static Blueprint-callable wrappers for LCMNav3D tactical primitives.

All functions are pure CPU and synchronous - they execute on the calling thread (Blueprint VM thread, BT service tick, EQS test evaluation, etc.) and return immediately. None of them dispatch GPU work or async jobs. Designed to be called at EQS-test frequency (tens to hundreds per query per frame) without budget concerns.

Functions

- `IsSegmentLOSClear` - Tests whether a straight line from Start to End is clear of unblocked SVO leaves for an agent of the given radius. Wraps the same Lazy Theta* neighbour-graph LOS walker used by any-angle pathfinding. Returns false if the SVO at Start cannot be resolved (out of bounds or unloaded chunk).
- `IsReachableSync` - Cheap reachability test - returns true iff a path from Start to Goal exists. Implemented by calling the underlying A* with early-out; faster than FindApexPathSync when you don't need the waypoint list.

- `IsPositionBlocked` - Returns true iff WorldPos resolves to a blocked SVO leaf (or to no leaf at all - out-of-bounds and unloaded chunks both report blocked by default for tactical-query safety). Pair with GetSweepPenaltyAt when you need to distinguish ”blocked” from ”merely risky”.
- `GetSweepPenaltyAt` - Reads the §P9.3 SweepPenalty byte at WorldPos. Higher values indicate the voxel was recently swept by a dynamic obstacle's predicted hull - a heuristic measure of ”how dangerous is this position right now?”. Returns 0 when the position resolves to a chunk that hasn't been loaded, when the voxel itself is blocked (treated as max danger by default in path costing - caller may want to treat blocked separately via Is Position Blocked), or when no dynamic obstacle has touched it. Value range: 0 (clear). 255 (maximum recent threat).
- `FindBestCover` - Wraps LcmSVOPathSolver::FindBestCover. Finds the best cover position for Agent-Location against ThreatLocation within SearchRadius (cm), preferring positions with line-of-sight broken to the threat. @param OutCoverLocation receives the best cover position when the function returns true. When the function returns false, this is left at ZeroVector.
- `FindApexPathSync` - Synchronous ”is there a path from Start to Goal” probe. Wraps the same A* core that FindPathAsync uses but runs on the calling thread for use inside EQS tests that need to score candidates by LCM Nav3D-path distance (NOT euclidean) before the agent commits. Returns true when a path was found; OutPath receives the waypoint sequence. Returns false when start or goal don't resolve to a loaded chunk, when their containing voxels are blocked, or when no path exists between them. Pay-as-you-call cost - typical short-range paths complete in <0.1 ms, long-range cross-chunk paths in 1-5 ms. If you only need the boolean ”is reachable?” answer without the waypoint sequence, prefer IsReachableSync (cheaper - bails as soon as goal is popped).