



Drop-In Water — Realistic URP System

Documentation · v1.0.0 · Tartine Studio

Tartine Studio · Version 1.0.0 · Unity 6000.3 and newer · Universal Render Pipeline 17 and newer

Drop-In Water brings oceans, lakes and rivers to Unity's Universal Render Pipeline. The waves are simulated with FFT from the wind, the swell and the depth, and the water is shaded from how light is absorbed and scattered in it. Floating objects feel the same waves the camera sees, and boats and splashes push the water with rings and wakes.

There is no setup: add an **Ocean** to a URP scene and press Play. There is no renderer feature to add and nothing to change in the URP asset. Gameplay code can ask for the height, the slope and the motion of the water anywhere. The answer comes immediately, costs microseconds, and is the same on every machine, so a multiplayer game only has to share its clock.

1. Key features

- **FFT ocean.** Wind waves come from the wind speed, direction and fetch (JONSWAP spectrum). A separate swell comes from distant weather. Four cascades cover every scale, from 1 km swells to centimetre ripples, with sharp crests and flat troughs.
- **Infinite surface.** A camera-centred level-of-detail mesh reaches the horizon, with no seams and no popping.
- **Lakes and rivers.** Lakes are wave water inside a box, at any height. The wind raises waves of the size a real lake of that length would get. Rivers follow a spline and flow downhill: they run faster on slopes and slower at the banks, and turn white in rapids. A river blends into the lake or sea it flows into or out of: it joins their level and fades into their water, whose waves calm under its current.
- **Shorelines.** Bake the terrain once. Waves then shrink and break into surf in shallow water, foam gathers along the shore, and dry land below sea level stays dry. Each wave runs up the beach with a small breaking bore and spray, then drains back, pulling its foam into streaks and leaving the sand wet and shiny.
- **Physically based shading.** The water's colour comes from depth-based absorption and scattering, and light shines through wave crests. The surface refracts and reflects the scene (screen-space reflections, reflection probes and the sky). The sun makes glints on the water and caustics on the seabed. Whitecaps appear where the waves break.
- **Underwater.** Fog and colour loss, a waterline across the lens, sun shafts shaped by shadows and caustics, and a seamless crossing through the surface.
- **Buoyancy. Floating Body** applies Archimedes' force and water drag; automated tests check the floating height against the exact Archimedes draft. Objects drift with the waves and down rivers. All floating bodies are computed in one Burst job per physics step.
- **Interaction waves.** Objects make rings and bow waves, and moving boats leave a true Kelvin wake. Every wavelength travels at its real speed (exact spectral solver), and foam appears where the water is churned.
- **Gameplay queries.** Height, normal and velocity of the water at any point, from a CPU copy of the waves. Queries are safe in FixedUpdate, run thousands at once in Burst jobs, and also work on a server with no GPU.
- **Multiplayer ready.** The waves are a function of the settings, the seed and the time. Every client computes the same sea.
- **Editor tools.** Every Tartine Studio feature is in one **Tools** menu. Inspectors show the material inline, river handles work in the Scene view, and debug views help tune the look.

2. Requirements and compatibility

Unity	6000.3 (Unity 6.3 LTS) or newer. Tested with 6000.3.8f1; imports and compiles cleanly in 6000.6.0f1.
Render pipeline	Universal Render Pipeline 17 or newer. Tested with URP 17.3 and the Forward+ renderer. Built-in RP and HDRP are not supported.
Platforms	Platforms with compute shaders: Windows, macOS, Linux and consoles. Tested on Windows with Direct3D 12. Mobile, WebGL and XR are not supported in this version.
URP settings	Nothing to set. The water asks each camera for the depth and opaque textures it needs.
Dependencies	Burst and Mathematics, which URP already installs.
Input	The water reads no input. The demo scripts work with both the Input System and the legacy Input Manager.
Physics	Buoyancy uses the built-in 3D physics (Rigidbody and colliders).

3. Package contents

```
TartineStudio/DropInWater/  
  Runtime/      Water bodies, wave simulation, buoyancy, interaction, rendering  
  Editor/       Inspectors, Tools menu, shoreline baker, river handles  
  Shaders/      Ocean, river, underwater, mask and spray shaders, two compute shaders  
  Materials/    M_Ocean, M_Lake, M_River, M_Spray  
  Textures/     Foam and spray textures  
  Demo/         Demo scene, its terrain and materials, and demo scripts  
  Documentation/ Markdown and PDF guide
```

4. Quick start

Open the demo

1. Open `Demo/DropInWater_Demo.unity` inside the package folder and press Play.
2. Hold the **right mouse button** and use **W A S D Q E** to fly (Shift = faster, mouse wheel = speed). Keys **1 - 8** jump to viewpoints: the bay, the beach, the pier, the boat, the lake, the river, under water and the open sea.
3. Press **B** to take the helm of the boat (W S throttle, A D rudder). Press B again to let the autopilot sail it in circles.

The demo is a bay with a beach, a pier and rocky headlands. A lake lies in the hills, and its river runs down to the sea. Buoys, crates, barrels, rafts, an anchored ship and logs float in all three waters.

Add water to your scene

1. **Tools > Tartine Studio > Drop-In Water > Create > Ocean.** The ocean sits at the height of its GameObject and reaches the horizon. It also adds a **Water Interaction** component, for ripples and wakes.
2. If the scene has a terrain, select the ocean and press **Bake Shoreline** in its **Shoreline** section (or **Tools > Tartine Studio > Drop-In Water > Bake Shoreline of Selected**). The waves now shrink and break along the coast, and run up the beaches.
3. Shape the sea in the ocean's Inspector: **Water** sets its colour and reflections, **Waves** the sea state (wind and swell), **Shoreline** what the coast does to it (swash, foam, spray). Each part that can be turned off has its own switch, and **Quality** holds what it costs (section 6).
4. For a lake or a river: **Tools > Tartine Studio > Drop-In Water > Create > Lake / River.** The new object appears at the centre of the Scene view. Size the lake's box to reach under the shores. For a river, place its points from the source to the mouth.

The same items are in **GameObject > Tartine Studio > Drop-In Water**, and in the Hierarchy's right-click menu, which parents the new water to the selection.

Make things float

Select objects with colliders and choose **Tools > Tartine Studio > Drop-In Water > Selection > Make Selected Float.** It adds a Rigidbody and a **Floating Body** (and a Box Collider when there is none). The objects now float on whatever water they are in, with a mass computed from their volume and density.

5. How it works

Part	What happens	Where
Waves	Each Ocean and Lake builds a wave spectrum from its settings and evolves it with FFTs, one per cascade.	GPU compute, every frame
Gameplay copy	A low-resolution copy of the same waves is computed for the current time. Height, normal and velocity queries read it.	CPU, Burst jobs
Surface	A grid centred on each camera, with detail that decreases with distance, is displaced by the waves. It reaches the horizon.	GPU, one draw per body and camera
Shading	Absorption, scattering, refraction, reflections, foam and caustics, from the camera's depth and opaque textures.	Transparent queue (Transparent-100; rivers right after, Transparent-99)
Underwater	Rendered when a camera is at or below a surface: a mask of the surface, then a full-screen pass.	GPU, only when needed
Buoyancy	Sample points inside each body's colliders are tested against the water every physics step.	CPU, one Burst job per water body
Interaction	A square of water around the camera is simulated with an exact spectral solver. The result is added to every surface inside it.	GPU compute

- **No setup in URP.** The water hooks into URP's camera callbacks. It draws itself for every camera and requests the depth and opaque textures for the cameras that see it.
- **Several bodies of water.** A scene can have one ocean and any number of lakes and rivers. `WaterBody.Find(position)` returns the water at a position (rivers first, then lakes, then the ocean). Floating objects, cameras and the underwater effect all use it.
- **Deterministic.** The waves depend only on the settings, the seed and `WaterClock.Now`. The same time gives the same waves on every machine, at any frame rate.

6. Water bodies

Every water body's Inspector starts with the Tartine Studio header and a **Documentation** button. Hover any field for a tooltip.

The Ocean and Lake Inspector starts with the **Material**, then is split into sections: **Quality** (what the water costs), **Water** (its look, with its caustics and whitecaps), **Waves** (the sea state), **Shoreline** (what the coast does to the water: the swash, the shore foam and the spray) and **Underwater**. Quality sets how much detail is spent on the water, the other sections what it looks like: a scene can be fitted to a platform without changing its look.

- Each part that can be turned off has a switch in its header, down to each effect inside a section. A part switched off keeps its settings, greyed out, and so does everything it holds: with the **Shoreline** off, its swash, foam and spray are off too.
- **All ON / All OFF** set every switch at once. All off leaves a flat, still water with only its look, to judge it; then switch the parts back on one by one.
- **Expand all / Collapse all** open or close every section. The sections start collapsed and stay as you leave them for the rest of the editor session; a section greyed out can still be opened.
- A section's header shows a short status on the right: the size of the simulation, the height of the waves, whether the shoreline is baked, the droplets of spray alive, or why a part does nothing (not baked, no waves, the swash's foam instead).

Ocean

An infinite sea at the height of its GameObject; X, Z and rotation do not matter. Put one per scene.

Section	Settings
Material	The material asset (shared by every water using it; section 7).
Quality	Wave simulation: Resolution and Cascade Count of the GPU waves, Largest Tile Size (longer than the longest waves), Gameplay Resolution of the CPU copy used by queries and buoyancy, Loop Period (the waves repeat exactly after it). Surface mesh: Vertex Spacing near the camera, Grid Resolution , Levels (each doubles the distance covered), Morph Range . Reflections: Screen Space Reflections , SSR Steps , SSR Max Distance (on the material). Underwater: Light Shaft Samples , Beam Shadow Samples (Point & Spot Lights > Shadows in the Water). See section 13.
Water	The material's colour and clarity and surface (section 7), and Edge Smoothness (the water depth over which the edge of the water fades in where it meets the ground and objects: higher = softer); the Reflections of this body: Skybox , Intensity , Blur , Sun Highlight (section 7); the Foam look shared by every foam of the body: Texture , Color , Tiling (the texture's red channel is the main pattern, green fine detail, blue large-scale density; tile it seamlessly, sRGB off)
Water > Caustics (switch)	Intensity , Fade Depth
Water > Whitecaps (switch)	Foam on the crests breaking in open water: Coverage (0.5 is calibrated on real seas), Lifetime
Water > Point & Spot Lights (switch)	Point and spot lights on the water, over and under the surface, with their shadows and cookies: Highlights (on the surface, reflected or coming through it) and Scattering (the light they spread in the water and on its foam, and the glow and beams around them under water); 1 = physically based. Shadows in the Water (off by default): objects cast shadows in that light and cut shadowed gaps in the beams. Section 7.
Waves (switch)	Wind: Speed (m/s at 10 m: 3 light air, 8 moderate breeze, 14 near gale, 25 storm), Direction (where it comes from, degrees clockwise from world +Z), Fetch (km of open water the wind blows over: short fetch = young, short, steep waves), Alignment (0 confused sea, 1 long-crested). Swell: Height (significant height, 0 = none), Period , Direction (where it comes from, degrees clockwise from world +Z), Spread . Shape: Choppiness (sharp crests), Water Depth used for the wave shape; under More : Peak Sharpness , Short Wave Cutoff , Time Scale , Seed . Off: a flat, still surface (the ripples of Water Interaction only), without surf, swash or spray; the simulation still runs, at the cost set in Quality.
Shoreline (switch)	The ground around the coast, baked from the terrain (section 8): Baked Data and the Bake Shoreline button, Wave Depth , Breaking Depth . Off: the water behaves as if there were no shore, as without baked data (and without its foam and spray).
Shoreline > Swash (switch)	The waves running up the beaches, with their bores, foam and wet sand (section 8)
Shoreline > Foam (switch)	Width (depth of water, m) and Intensity of the foam along shores and around objects; Surf , the foam of the waves breaking in shallow water. With the Swash on, its own foam replaces all of it: the Foam is greyed out.
Shoreline > Spray Particles (switch)	Droplets thrown up by the bores of the swash (section 8).
Underwater (switch)	Waterline , Light Shafts and their Light Shaft Intensity

Lake

Wave water inside a box, at any height: lakes, ponds, pools, flooded caves. It has the Ocean's settings, plus a **Lake** section under the Material:

Setting	Use
Size	Width and length of the box (m), centred on the object and rotated with its Y rotation. Make it reach under the shores: the terrain hides the rest.
Depth	How far below the surface the lake counts, for cameras and floating objects.
Edge Fade	Distance over which the waves fade out before the edges of the box.
Fetch From Size	On by default: the wind blows over the length of the lake, so a pond ripples and a 5 km lake gets real waves (the wind's Fetch is then greyed out).

The default material, M_Lake, is fresh water: greener, with shorter visibility than the sea.

River

A flowing ribbon of water along a spline, from the source to the mouth. Its Inspector has an **Appearance** foldout (the reflections, the point and spot lights and the material, section 7) above its settings.

- **Scene view.** Move the points with the handles, drag the blue dots to set the width, and **Shift + click** on the ground to add a point at the end.
- **Fit Width To Terrain** (Inspector button, or **Tools > Tartine Studio > Drop-In Water > Fit Selected River to Terrain**) widens or narrows each point until both edges are under the banks. **Reverse Flow** swaps the source and the mouth.
- **Flow.** The water flows along the spline. Its speed is Base Speed + Slope Speed x sqrt(slope), up to Max Speed. It is fastest in the middle and slower at the banks (**Bank Speed**). Ripples and foam are carried by the flow, and the water turns white where it is fast (material: Rapids Speed, Rapids Foam).
- **Never Uphill.** A point higher than the one before it is lowered, so the water never flows uphill.
- **Shape.** **Depth** of the channel, **Segment Length** and **Cross Segments** of the mesh.
- **Mouth and Source.** Where the river flows into a lake or the sea, or out of one, it blends into their water (**Blend With Water**, on by default). Over its last **Blend Length** metres (24 by default) the river first joins their water level, then fades out into their water, from its sides first, like a tongue of river water spreading out: its colour and ripples give way to theirs, and their waves, calmed under its current, come back. Under water it blends the same way: swimming from the river into the lake or sea, the river's colour, clarity and light turn into theirs as its water gives way to theirs, without a jump. Nothing to set: the river finds the Ocean or Lake its first and last points lie in, when they lie over their water from 1.5 m under their level to 1 m above it. Where the river has not started to fade, the lake or sea is not drawn there, nor further up the river wherever their water would show inside it (a channel dug below sea level, from their shoreline data; without it, down to the river's **Depth**), so a lake's box may reach over the river where it leaves the lake.
- **Surface.** **Ripples** are small wind waves carried by the flow, with their own small simulation. Underwater and Caustics work as on the ocean.
- Queries return the flow velocity, so floating objects drift downstream on their own. Outside the river there is no water: `SampleHeight` returns -1000000.

To connect a river to a lake or the sea, start or end it a little inside their water, at their level: its first or last Blend Length metres fade into them. At the mouth, end the river about half a Blend Length past the point where it reaches their level. The demo's river blends this way into the sea, and out of the lake.

7. Appearance: the water material

The Ocean and Lake use the **Tartine Studio/Drop-In Water/Wave Water** shader; rivers use **River**. Edit the material in the **Water** section of the Ocean and Lake Inspector (its reflection quality in **Quality**), in the River's **Appearance** foldout, or on the material asset. Material settings are shared by every body using that material; **Reflection Intensity**, **Reflection Blur** and

Sun Reflection Intensity are per component (on the Ocean and Lake: **Water > Reflections > Intensity, Blur and Sun Highlight**). The Ocean and Lake have their shore and foam settings on the component too: **Water > Edge Smoothness** and **Foam look**, and **Shoreline > Foam**; the material's are for rivers.

Group	Settings
Water Body	Scattering Color (light coming back from inside the water), Absorption Color (colour left after travelling through the water), Absorption Distance (visibility, m), Crest Scattering (light through thin wave crests)
Surface	Extra Roughness , Refraction Strength , Refraction Smoothing (higher = steadier refraction that follows only the larger waves), Rain Ripples (the rain's rings, with a sky that publishes its rain; 0 = none), Shore Fade (soft edge where the water meets the ground; rivers only)
Reflections	Screen Space Reflections on/off, SSR Steps , SSR Max Distance . Beyond them, reflection probes and the sky are reflected.
Foam (rivers; the Ocean and Lake use their Foam look and Shoreline > Foam)	Foam Texture , Foam Color , Foam Tiling , Shore Foam Width , Shore Foam Intensity
Flow (River)	Flow Cycle , Ripple Strength , Rapids Speed , Rapids Foam

- **Skybox** (component, on by default; the River's **Skybox Reflection**): the water reflects the skybox as it is now, and takes its ambient light from it. It captures the sky itself, again whenever the sky material, any of its settings or the sun changes (at most ten times a second while it keeps changing, as in a day-night cycle), so it follows the sky without generating the lighting. The ambient light (on the water, its foam, its spray and under it) is the light the captured sky gives, times the Lighting window's ambient **Intensity Multiplier**, when the scene's **Environment Lighting Source** is Skybox; with Gradient or Color, the water keeps those. Off: the water reflects the scene's reflection probes instead, as other objects do (the environment reflection generated in the Lighting window, made with the sky and sun of that moment, or your local probes), and takes the scene's ambient light. Either way, what is on screen is reflected by the screen-space reflections on top. A sky that draws its own environment every frame can hand it to the water instead (Drop-In Sky & Weather, sold separately, does with its Environment Lighting on): the water then reflects that sky as it is, its clouds as they move, without the sun's and the moon's disks (their lights make the highlights), and captures nothing itself.
- **Fog**. With a sky that publishes its fog (Drop-In Sky & Weather, sold separately), the water is fogged by it like the rest of the scene, mist and shafts of light included, and so are its spray and what it reflects; otherwise it uses Unity's fog (Lighting window, Environment > Fog). Nothing to set.
- **Rain**. With a sky that publishes its rain (Drop-In Sky & Weather, sold separately), rings spread where the drops fall on the water. They are drawn on the surface itself, so they ride the waves back and forth as the foam does, and drift downstream with a river's flow. Falling snow rings the water too, more softly. They show near the camera, as far as the sky's Splash Distance. Farther away, where they are too small to see, they roughen the surface: duller reflections and a broader glint of the sun. The material's **Rain Ripples** sets how strong they are (1 = natural). Without such a sky, nothing changes. The water also publishes where its surface is around the camera (`_TartineWaterHeights`: its height seen from above, at rest), so such a sky's rain and snow stop at the surface instead of falling through to the bottom, and splash on it.
- **Point & Spot Lights** (component, on by default; the Ocean's and Lake's **Water > Point & Spot Lights**, the River's **Appearance > Point & Spot Lights**): URP's point and spot lights light the water like the sun does, with their shadows and cookies: their highlights on the waves, their light coming through the surface (a lamp in the water seen from above as a refracted point of light, a lamp above the water seen from under it through the surface, with total internal reflection beyond the critical angle), the foam they light, and the light they spread in the water under the surface, from a lamp above it or in it, absorbed along the way. On beaches the swash takes them too: its sheet and bores like the sea, its foam, and the film of water it leaves on the sand, which reflects a lamp (and the sun) as sharply as it is smooth (a bright streak on the sand just after the backwash). Under water, the water glows around them, fading with the absorption, and a spot light shows as a beam, with its cone and its cookie's pattern. **Highlights** and **Scattering** scale these (1 = physically based). The glow seen under water needs the Forward+ renderer (the URP default; Deferred+ too); the rest works with every renderer, within URP's per-object light limit with Forward. Off: the sun only, cheaper with many lights.

- The light's shadows always apply to its highlights and to the foam. **Shadows in the Water** (off by default) applies them in the water too: objects between a light and the water shadow the light it spreads there, and cut shadowed gaps in its glow and beam seen under water, such as a lamp shining through a grate or a hull in front of a diver's spot light. Each shadowed light's beam is then sampled at **Beam Shadow Samples** points (Quality, 4 - 32, 12 by default) with a shadow lookup each, spaced by its light and jittered every frame like the sun shafts: more samples give sharper, steadier shadow edges in the beams, at a higher cost. It only concerns lights that cast shadows: set the light's **Shadow Type**, and enable **Lighting > Additional Lights > Cast Shadows** in the URP asset.
- A spot light's beam is cut exactly to its cone before it is sampled, so its edges stay clean whatever the number of samples.
- **Reflection Intensity** (component, 0 - 2) scales the reflections of the sky and the scene (reflection probes and screen-space reflections) on that body only, on the wet sand of its swash and on its foam (the foam reflects the sky like the water, blurred: lit by neither the sun nor the ambient light, it still shows what the water around it reflects, instead of turning black). 1 is physically based: weak looking down, strong at grazing angles. 0 removes them, so only what is under the water shows; above 1 they get stronger. The sun highlight is not affected.
- **Reflection Blur** (component, 0 - 1) blurs the reflections of the sky and the scene on that body, on top of what its ripples blur, as on a rougher surface. 0 is physically based; towards 1 the reflections turn into a smooth blur (the screen-space reflections hand over to the reflection probe). The sun highlight is not affected. On the Ocean and Lake, the swash and the wet sand have their own: **Shoreline > Swash > Reflection Blur**.
- **Sun Reflection Intensity** (component, 0 - 10) scales the direct sun highlight of that body only: on the water, and on the Ocean and Lake on the film of water the swash leaves on the sand, where the sun's shadows cut it. 1 keeps the physically based glint; 0 removes it. The sun light, the sky reflection and the material are unchanged. On the wet sand the glint, like the reflections, is as sharp as the film is smooth: a mirror on the water still standing after the backwash, spreading out and dimming where the film thins out into damp sand.
- **The sun's cookie**. The water follows the cookie of the scene's main light, as the objects around it do: cloud shadows a sky asset casts through the sun's cookie, or a cookie texture of your own, darken its glints, the light scattered in the water, its foam, the glint on the wet sand, the caustics and, under water, the light shafts and the water's glow, and its spray (which also takes the sun's shadows). Nothing to set; without a cookie, nothing changes.
- Colours are picked in sRGB like every colour field. The shader works in linear space.
- To give two bodies different looks, duplicate the material and assign the copy to one of them.

8. Shorelines

Select an Ocean or a Lake and press **Bake Shoreline** in its **Shoreline** section, or choose **Tools > Tartine Studio > Drop-In Water > Bake Shoreline of Selected**, or **Bake All Shorelines in Scene**. The component's context menu has **Bake Shoreline** as well. The ground heights of the terrains around the water are stored in an asset next to the scene (`<Scene>/Shoreline_<name>.asset`). With them:

- waves shrink in shallow water and stay off dry land, on the GPU and in the gameplay queries alike (**Wave Depth**: depth, in wave heights, where they keep 63 %);
- waves break into surf where the water gets shallow (**Breaking Depth**, and **Shoreline > Foam > Surf**);
- foam lines the shore (**Shoreline > Foam**), and the water fades softly against the ground (**Water > Edge Smoothness**, with or without baked data; with the swash, the sea fades into the wet sand or the thin sheet, over at least half the depth the swash starts at, and follows the backwash down, so the two meet without a line at the still water line);
- the ocean no longer counts as water on dry land below sea level: no underwater view in a valley, no buoyancy on a beach.

Beyond the baked area the ground is unknown: past an edge that is under water, the sea floor is taken as going on down (1 m every 10 m), so the open ocean keeps its full waves; past an edge on land, it stays land. **Wave Depth** therefore only matters where the baked water is shallow compared with the waves: if it changes the waves everywhere, the sea floor of the baked area is shallow everywhere.

The **Shoreline** switch turns all of this off at once, without losing the baked data: the water then behaves as if there were no shore. Bake again after changing the terrain. From a script, `ShorelineBaker.Bake(water, area, texelSize, includeColliders)` does the same (Editor only).

Swash: waves running up the beach

With baked data, each wave is followed from the breaker line to the top of the beach as one body of water. It breaks where the water gets as shallow as **Breaking Depth** and rolls in as a bore: a real raised front of white water (its amount is the swash's **Foam > Bore Clumps**, times its **Intensity**), moving at the speed of waves in shallow water and leaving foam behind. It reaches the still water line exactly when its run-up starts, and a strong bore makes a long run-up. The front climbs the beach, slowing down, its bore collapsing into a thin sheet that refracts the ground under it. Then the water drains back faster, pulling the foam left on the sand into streaks down the slope and drawing the sea down below the still water line until the next wave floods it back, and the sand it leaves stays dark and shiny for a while. Timing, strength and the fingers of the front vary along the shore. With the swash on, its foam replaces the Shoreline's (**Shoreline > Foam** is greyed out): the bores replace the surf foam drawn on the crests of the waves, and the swash's water and foam replace the line of foam along the shore, including around objects standing in the water.

The **Swash** (in the Shoreline section) has a switch in its header, on by default; the effect needs the baked data and the waves. Its parts (**Bore, Sheet, Foam, Wet Sand**, under **Parts**) follow; the Bore, the Foam and the Wet Sand have their own switch too, so each can be judged on its own.

Waves	Use
Run Up Height	How high the waves run up above the still water level (m). About $0.6 \times$ the significant wave height is typical of sandy and pebble beaches. It alone sets how far the swash goes: it starts a little below the still water line ($0.45 \times$ the run-up) and climbs to the run-up. The height of the waves (wind, swell) does not change it; the bores and the surf follow the waves. (Scenes saved with the former Run Up , in significant wave heights, are converted when first drawn, to the same height for the waves they have then.)
Period	Time between two waves running up (s). 0 = the peak period of the waves.
Uprush Share	Part of each wave's cycle spent running up; the rest is the backwash.
Reflection Blur	Blur of the reflections on the swash and on the wet sand, independent of the sea's (section 7).
Variation > Height	How much the waves differ: each runs up (and breaks) up to this much higher or lower ($0.35 = \pm 35\%$).
Wave direction > Follow Wave Direction, Lee Side	With the switch on (default), the swash follows the direction of the waves; Lee Side is how much of it is left on shores facing away from them, such as the far side of an island (0 = none, 1 = the same all around). Off: the same swash on every shore, at times that vary along it (Variation > Timing). The waves come from the swell's Direction (the wind's when the swell is less than half as high as the wind sea): shores facing them get the whole run-up, bores, foam and wet sand, less and less round the sides, down to the Lee Side behind. How far round they still reach follows their spread of directions (Swell Spread , or the wind's Alignment). They also reach each stretch of shore when their crests get there, so on an oblique beach the waves run up in sequence along it, and they reach the far side of an island last.
Variation > Timing	How much the time the waves reach the beach varies along the shore, in wave periods. 0 = the whole shore at once.
Variation > Scale	Length of shore over which the timing changes (m); the height changes over half of it.

Bore (switch: off = the waves break as ordinary surf, with foam on their crests)	Use
Height	Height of the bore where it turns into the swash: 0 = none, 1 = the largest ($0.02 \times$ the significant wave height). Taller further out; a bore never gets taller than about $0.8 \times$ the depth of the water (it would break). It only changes how tall the bore is, not how far the waves go. The bore is a smooth wave: its face spans a few cells of the surface mesh (Vertex Spacing), so the mesh and its shading draw it the same at every distance, without steps. (Scenes saved with the former Bore Height , in wave heights, are converted when loaded, to the same bore up to the largest.)
Duration	How long the water stays raised behind the front, as a fraction of the period.
White Water / Lace	How long the white water, then the lace it leaves, last (s). White Water 0 = none: the bore brings only lace.

Sheet	Use
Front Depth, Deepening, Max Depth	Depth of the water behind its front (cm), how much deeper per metre behind it (cm/m), and at most (cm).
Backwash Thinning	How much thinner the sheet gets while it drains.
Swash Smoothness (min, max)	Softness of the front of the swash: width of its thin tip (m), where the water thins to nothing and the sand shows through. The front uses the lowest value while it moves fast (just after the bore reaches the beach, and at the end of the backwash) and the highest where it slows down and turns at the top of the beach, so it never switches from sharp to soft. Varies along the shore. 0, 0 = a sharp front.
Infiltration, Infiltration Edge	In the backwash the water also soaks into the sand: the sheet turns transparent as it comes down, completely when its front reaches the still water line, leaving shiny wet sand. Its thin upper edge goes first, over a width that grows as it drains: the Swash Smoothness plus up to the Infiltration Edge (m). 0 = the water only drains back into the sea.
Slope Follow	How much the surface follows the slope of the sand. At grazing angles every degree of tilt lifts the reflection by two degrees: at 1 the sheet reflects the sky above what the sea next to it reflects.
Edge Fade	Depth over which the sheet turns clear at its very edge (mm), where it carries the water on up the sand. Its surface still reflects like the rest of the water, however thin.
Fingers, Finger Length, Finger Size	Tongues of thin water running ahead of the front: on/off, how far (m), how wide (m).

Foam (switch)	Use
Intensity	Overall amount of swash foam.
Pattern Size	Size of the holes in the foam lace (m); the patches of foam are some four times bigger.
Front Line, Front Line Width	The solid white line at the front of the uprush, and its width (m) as the wave starts up the beach (40 % of it at the top).
Bore Clumps, Clumps Width	White water behind the front, and how far behind it reaches (m). Bore Clumps (times the Intensity) is also the white water of the bores rolling in over the surf zone, or of the surf on the crests with the Bore off. It appears as the wave brings it (rolling in with the bore) and lasts the bores' White Water time (Bore settings); the foam builds up over a short distance behind the front, so its leading edge is soft.
Lace, Lace Time	Foam lace left on the sheet, and how long it takes to break up (s).
Backwash Streaks, Streak Stretch	How much the draining backwash pulls the foam into streaks down the slope, and how much longer than wide it gets at most. It stretches from the top of the run-up, slowly at first, faster as the backwash speeds up.
Backwash Drift	How far the backwash carries the foam back down: 1 = along with the retreating front, 0 = it stays where the uprush left it and breaks up there.
Bore Carry, Absorption	In the surf zone, how far the passing bores carry the foam on the water towards the beach (1 = with the water they move), and how far ahead of a bore its face draws in and swallows the old foam (m). The foam behind a bore is new foam it brings, building up just behind its front, and only where the waves have broken (shallower than Breaking Depth).
Swash Mark, Swash Mark Time	The thin line of foam each wave leaves on the sand at its highest reach, and how long it stays (s).

The foam is procedural and never repeats: a lace of bubbles around holes of clear water, with small bubbles where it is dense and larger holes where it thins. It is carried by the water, up the beach and back down, and lies on top of the water and the wet sand, so the white line stays at the very front of the thin tip of the swash.

Wet Sand (switch)	Use
Intensity	Overall strength of the wet sand.
Darkening	How much darker wet sand is.
Wetting Time	How long the sand takes to get wet once the water reaches it (s): the wetness comes in gradually where the thin tip of the water shows the sand.
Drying Time	How long the sand takes to dry once the water has left it (s), down to the Persistent Wetness.
Edge Softness	Width over which the wetness fades out below the highest point each wave reached (m): a soft upper edge that follows the shape of the water.
Persistent Wetness	Dampness the sand most waves reach keeps however long it has dried (fading out towards the top of the beach). 0 = it dries completely between waves.
Reflection	Reflection on the wet sand.
Sheen, Sheen Time	Shine of the water still standing on the sand just after the backwash, and how long it lasts (s). It reflects the scene like the water it comes from.

The wetness comes from the last four waves: each wets the sand its water covered, from the moment it got there, and starts drying from the moment it left, so it never switches on or off at once.

The swash is drawn per pixel on the ground as it is actually rendered (terrain, pebbles, rocks, a character's feet): over beaches the water mesh is lifted just above the highest water, and each pixel decides between the swash sheet, the sea, wet sand and dry sand. It is visual: the gameplay queries and buoyancy see the still water level there. Like the waves, it depends only on the position and the time, so it is identical on every machine.

Spray Particles (Shoreline > Spray Particles; on by default on a new Ocean) throws spray droplets from the bores near the camera, as they roll in over the surf zone and as they climb the beach, using the same model to find the fronts, in Play mode around the Main Camera, and outside Play mode around the Scene view's camera (**Preview In Edit Mode;** it shows as the Scene view redraws: turn on **Always Refresh** to see it move). It says why no spray is thrown, if so (no baked data, waves, shoreline, swash or bores off, no Main Camera), and shows the droplets alive. It is kept in a **Shore Splashes** component that the switch adds to the water when needed; the component is hidden (everything is set in the water's Inspector) and is removed with the water. From a script: `GetComponent<ShoreSplashes>()`. Ranges (lowest, highest) have a slider; each droplet takes a random value in them.

Spray Particles	Settings
Emission	Only From Foam (droplets only where the water is white, as drawn: the spray comes out of the foam's patches), Rate (droplets per second along 100 m of front from the bores climbing the beach, at full strength, as the bore reaches the beach, where the foam is dense; less as the bore collapses and where the foam thins), Distance from the camera, Max Particles , Burst (droplets thrown together), Min Bore Strength (the bore collapses as it climbs: no spray below it), Spread Along the front and Spread Behind it (m), Launch Height above the water (in bore heights).
Surf Zone (switch)	Spray of the bores rolling in over the surf zone, from where the waves break to the beach: Rate (counted as the Emission's: droplets per second along 100 m of bore front at full strength, a bore at least as tall as when it reaches the beach, where the foam is dense; the same value gives the same spray on the beach and over the surf zone). It starts on the water as it is there (waves included). The other Emission settings, the Motion and the Appearance apply to it too.
Motion	Height of the throw (in bore heights), Upward Speed and Forward Speed (times the speed reaching that height, and times the speed of the front), Max Forward Speed , Scatter (random sideways speed), Gravity , Drag , Lifetime (s).
Appearance	Size (m, mostly small), Size Over Lifetime (curve), Color (tint), Opacity , Color Over Lifetime (gradient: fades in, then out), Stretch into streaks along the motion (Stretch With Speed , Stretch Length ; off: round droplets).
Material	The droplets' material: texture, base colour, and Soft Distance where they meet the scene.

The spray is tested against the opaque scene only, so the lifted water mesh never hides it. From a script, call `ApplySettings()` after changing the curves, gradient, gravity, drag or stretch.

9. Floating objects and boats

Add **Floating Body** to an object with a Rigidbody and colliders (or use **Make Selected Float**). It floats on the water it is in, or on the one set in **Water**. The volume of its colliders is sampled on a grid. At each point under the surface, every physics step applies:

- buoyancy: water density x g x submerged volume (Archimedes);
- drag against the motion relative to the water, so waves and currents push floating objects the way real ones do.

Setting	Use
Water	The water to float on. Empty = whatever water the body is in (river, lake or ocean).
Samples Per Axis	Sampling grid per side of the colliders' box. More points = more precise pitch and roll, higher cost.
Water Density	1025 = sea water, 1000 = fresh water.
Mass From Density / Density	Sets the Rigidbody mass from this density and the colliders' volume. 500 = light wood, 900 = ice; above the water density the object sinks. Turn it off to keep your own mass.
Water Drag / Water Angular Drag	How strongly the water damps motion and rotation.
Drag Per Axis	Drag multiplier along the object's local axes. Hulls: (1, 1, 0.1), so they glide forward and resist sliding sideways.
Make Waves / Wave Strength	Pushes the water where the body moves (needs a Water Interaction, see section 10).

Tips:

- Objects float the way their shape and weight make them float: a uniform tall cylinder lies on its side, and a light cube floats edge up. To keep something upright, lower its centre of mass (Rigidbody > Automatic Center Of Mass off, Center Of Mass below the middle), as ballast does on real buoys and boats.
- Box, sphere and capsule colliders are sampled exactly. Mesh colliders must be convex to be sampled precisely (a non-convex mesh collider counts as its full box).
- Call `BuildSamplePoints()` after changing the colliders at runtime.

Demo scripts (in `Demo/Scripts`, free to reuse):

Component	Use
Boat Controller	Drives a floating boat: W S throttle, A D rudder, or set <code>throttle</code> and <code>rudder</code> from code (Read Input off). The propeller only pushes while under water, and the rudder only steers when the boat moves.
Boat Circle Autopilot	Sails a Boat Controller around a circle (Centre, Radius, Throttle). Disable it to drive by hand again.
Demo Camera	The fly-through camera of the demo, with viewpoints and a key to take the helm.
Network Water Clock	Shares a server clock between players (see section 11).

10. Interaction: ripples and wakes

A **Water Interaction** component simulates a square of water around the camera, or around its **Target**. Creating an Ocean adds one. Floating Bodies (with **Make Waves**) and **Water Interactor** spheres push the water, and the resulting waves are added to every water surface inside the square.

- The waves are evolved exactly: every wavelength travels at its true deep-water speed. Objects dropped in the water make rings, and moving boats leave a real Kelvin wake at any speed.
- Only what objects do relative to the water makes waves. An object riding the swell makes none; one moving through the water, bobbing or dropped in does.
- Foam appears where objects churn the water fast (a planing hull, a splash) and where the waves get steep.

Setting	Use
Target	Centre of the simulated square. Empty = the main camera.
Resolution / Size	Texels and metres per side of the square. Finer ripples need more texels per metre.
Damping	How fast the waves die out.
Strength	Height of the waves objects make. 1 = the water they really displace.
Foam Lifetime, Foam From Objects, Foam From Steepness	Foam of the interaction waves.

Water Interactor is a sphere (**Radius, Strength**) that pushes the water without floating: a swimmer, a paddle, a character's legs, a projectile. **Tools > Tartine Studio > Drop-In Water > Selection > Make Selected Push the Water** adds one sized to the object. In Play mode the Water Interaction Inspector shows the simulated square and how many objects push the water.

The interaction waves are visual: gameplay queries and buoyancy use the deterministic waves only.

11. Gameplay: queries and multiplayer

Querying the water

Every water body (Ocean, Lake, River) answers the same queries. The waves are read from a CPU copy computed for the current frame, so the queries answer immediately and cost microseconds. They work in FixedUpdate and on a server with no GPU.

Call	Returns
<code>water.SampleHeight(position)</code>	height of the surface above/below a point
<code>water.IsUnderwater(position)</code>	true below the surface
<code>water.SampleSurface(position)</code>	height, normal and velocity of the water (orbital motion, river flow)
<code>water.ScheduleHeightQueries(positions, heights)</code>	the same for thousands of points, in a Burst job
<code>water.ScheduleSurfaceQueries(positions, samples)</code>	height, normal, velocity in a Burst job
<code>water.GetUnderwaterState(camera)</code>	above water / crossing the surface / underwater
<code>ocean.GetWaveField()</code>	raw access for your own Burst jobs (call <code>ocean.RegisterReader(handle)</code>)
<code>river.GetRiverField()</code> , <code>river.Locate(position)</code>	river data for Burst jobs, and where a point is across the river
<code>WaterBody.Find(position)</code> , <code>WaterBody.TrySampleSurface(position, out sample)</code>	the water at a position (river, lake or ocean)

```
using TartineStudio.DropIn.Water;

if (WaterBody.TrySampleSurface(transform.position, out var water) && transform.position.y < water.height)
    Debug.Log($"Under water, current {water.velocity}");
```

At a river's mouth or source the queries blend as the surface does: the current fades out into the lake or sea and their waves come back, with no step where the river ends.

Complete the jobs within the frame. `Ocean.Active` is the ocean enabled last. In `FixedUpdate` the queries return the waves at the time of the physics step, not of the frame, so physics gives the same result at any frame rate.

Multiplayer

The waves are a function of the settings, the seed and the time: every client computes the same sea from the same time, so nothing about the water needs to be sent. Share the clock:

```
WaterClock.TimeSource = () => NetworkManager.Singleton.ServerTime.Time; // Netcode for GameObjects
```

The **Network Water Clock** demo component does this with smoothing. Call `Synchronize(serverTime)` when your transport gives you the server time, and the waves catch up without jumping.

For bit-identical physics between machines (lockstep, rollback), step the physics at the same times on every machine and make `WaterClock.Now` return the time of the step (e.g. `tick * tickInterval`). The waves repeat exactly after **Loop Period** seconds, so a long-running server keeps full float precision. A server built with `-nographics` still computes the waves for queries and buoyancy.

12. Tools menu and debug views

Tools > Tartine Studio > Drop-In Water >	Does
Create > Ocean / Lake / River / Water Interaction	Creates the water. Lakes and rivers are placed at the centre of the Scene view. It warns before a second ocean and selects the existing Water Interaction if there is one.
Selection > Make Selected Float	Adds a Rigidbody and a Floating Body (and a Box Collider if there is no collider).
Selection > Make Selected Push the Water	Adds a Water Interactor sized to the object.
Bake Shoreline of Selected / Bake All Shorelines in Scene	Bakes the terrain under the ocean and lakes (section 8).
Fit Selected River to Terrain	Fits each river point's width to the banks.
Debug View > ...	Replaces the shading of every water surface with a diagnostic view (below).
Documentation / Online Documentation / Contact Support	This guide, the web version, and an email to support.

The debug views show what the refraction does. From a script: `WaterDebug.View = WaterDebugView.RefractionScale`.

The refraction follows the bent view ray to the bottom. Where that point is hidden (behind an object in front of the water, or off screen), its distance is estimated, so the absorption stays continuous; the colour is then read closer to the pixel, smoothly, from what the screen shows under the water.

- **Refraction Scale.** How much of the refraction offset the colour lookup uses. White = all, darker = pulled towards the pixel near a screen edge or an object in front of the water.
- **Refraction Offset.** Red and green show the offset of the colour lookup (0.5 = none).
- **Refraction Grid.** A checkerboard seen through the refraction. It should bend smoothly, never shatter.
- **Refraction Search.** How much an object in front of the water limits the colour lookup.
- **Refraction Foreground.** Where the bottom the ray reaches is hidden: white = behind an object in front of the water, grey = off screen (both estimated).
- **Refraction Distance.** Length of the light's path through the water (white = 16 m). It should stay continuous around objects.
- **Swash.** On beaches: red = swash water, green = wet sand, blue = foam; dark blue = the sea elsewhere.

13. Performance

Cost depends on the screen coverage of the water, the simulation settings, the number of water bodies and cameras, and whether the camera is under water. Profile in your own scene and on your target hardware; the Profiler and the Frame Debugger show the water's work as `DropInWater.WaveSimulation`, `DropInWater.Mask` and `DropInWater.Underwater`.

Cost	Setting
Wave simulation (GPU)	One FFT per cascade and per Ocean or Lake: Quality > Resolution (256 by default) and Cascade Count (4). Small, calm lakes can often use 128 and 3 cascades. With Caustics off, the pass that prepares them is skipped too.
Gameplay copy (CPU)	Quality > Gameplay Resolution (64 by default), computed in a Burst job. It sets the precision of queries and buoyancy.
Surface mesh (GPU)	Quality > Grid Resolution and Levels set the vertex count; Vertex Spacing sets the detail near the camera. Levels beyond the camera's far clipping plane are not drawn.
Reflections (GPU)	SSR Steps per pixel. Turn Screen Space Reflections off for probe and sky reflections only.
Sky reflection (GPU)	With Skybox on: the skybox drawn into a 128 x 128 cubemap and filtered, with its ambient light, only when it changes (at most ten times a second). Nothing while the sky stays the same. With a sky that hands over its own environment, only the ambient light (a small cubemap, ten times a second).
Underwater (GPU)	Only while a camera is under water: Light Shaft Samples per pixel, or Light Shafts off.
Sky fog (GPU)	Only with a sky that publishes its fog: one texture read per water pixel, two more where a screen-space reflection hits.
Point & spot lights (GPU)	Per water pixel, the lights URP gives it (Forward+: those around it); under water, every visible light whose range the view ray crosses, in closed form: 2 evaluations per light, 8 for a spot light or a light with a cookie (its beam). With Shadows in the Water : a shadow lookup at the point under the surface per light, and under water Beam Shadow Samples shadow lookups along the beam of each light that casts shadows, per pixel. Switch Water > Point & Spot Lights off if the water does not need them.
Interaction (GPU)	Resolution of the simulated square (256 by default). Remove the component if nothing pushes the water.
Buoyancy (CPU)	Samples Per Axis cubed points per floating body, in one Burst job per water body.
Rivers	The river mesh (Segment Length , Cross Segments) and the ripple simulation of each river (on the GPU only: queries follow the flow, not the ripples). Where a river blends into a lake or the sea, both are drawn over its last Blend Length metres, and the lake or sea reads a small map of the river (128 x 128, made when the river changes) near its end.
Swash (GPU)	Only on beaches, within the swash zone: a depth read, a few noise and foam samples per pixel. Switch the Swash off to remove it.
Spray Particles (CPU)	A few hundred model evaluations per second and one particle system; Rate and Max Particles .

14. Limitations

- URP only. Compute shaders are required: mobile, WebGL and XR are not supported in this version.
- One ocean per scene. Lakes are boxes, rotated around Y only.
- Interaction waves are visual: they do not move floating bodies and are not returned by the queries.

- Water is not removed inside hulls: an open boat that sits low shows water inside it below the waterline.
- Rivers have no dedicated waterfall rendering: steep drops are drawn as fast white water.
- Rivers blend into lakes and the sea, not into other rivers. Up to four river ends blend into one lake or sea at a time: the nearest to the camera.
- Shorelines are baked: bake again after changing the terrain. Only terrains are baked by the menu items.
- The swash and the bores are visual: queries and buoyancy do not see them. Over beaches the water mesh is lifted above the highest swash water and writes its depth there: transparent objects drawn after the water, very close to the sand in the swash zone, can be hidden (the included spray is not).
- Refraction and screen-space reflections only show what is on screen, and transparent objects behind the water are not refracted.

15. Troubleshooting

- **The water is pink.** URP is not the active render pipeline (**Project Settings > Graphics and Quality**).
- **The water reflects another sky** (a bright day sky at dusk, or the sky before you changed it). With **Water > Reflections > Skybox** off, the water reflects the environment reflection generated in the Lighting window, which keeps the sky and sun it was made with: turn Skybox on, or press **Generate Lighting** again. The ambient light of the scene is generated the same way: with Skybox on the water takes its own from the sky, but the other objects (and the ground seen through shallow water) keep the generated one until you generate the lighting again.
- **The water is black or reflects black.** With Skybox off, the scene has no environment reflection. Open **Window > Rendering > Lighting** and press **Generate Lighting**, with Environment Reflections set to Skybox.
- **The sea shows on dry land below sea level, or in a valley.** Bake the shoreline (section 8), and bake again after changing the terrain.
- **The waves are too big on a small lake.** Keep **Fetch From Size** on, or lower the wind speed.
- **An object sinks, jumps or spins.** Check that its density is below the water density, that its mesh colliders are convex, and that the colliders match its shape. Call `BuildSamplePoints()` after changing colliders at runtime. To keep it upright, lower the centre of mass.
- **There are no ripples or wakes.** The scene needs a **Water Interaction**, the object must be inside its square (around the camera or Target), and it needs **Make Waves** (Floating Body) or a **Water Interactor**.
- **The seabed shimmers through the water when the camera moves.** Raise **Refraction Smoothing** on the material (default 8): the refraction then follows the larger waves only.
- **Shadows of tall objects disappear close to the camera.** Piers, masts or cliffs lose part of their shadow near the camera, cut along a curved line. This is URP's default **Shadow Near Plane** of 0.2 on the directional light: raise **Light > Shadows > Near Plane** to 3 - 10.
- **A lake shows water beyond its basin.** Its box reaches ground lower than the water: shrink **Size**, or raise the ground there.
- **A river flows the wrong way.** Press **Reverse Flow** in the River Inspector.
- **A river ends in a hard line in a lake or the sea.** Turn on **Mouth and Source > Blend With Water**, and make its first or last point lie over their water, from 1.5 m under their level to 1 m above it: its last **Blend Length** metres then fade into them. A river ending higher above them (a waterfall) is left as it is.

16. Support

Tartine Studio · support@tartine-studio.dev · tartine-studio.dev

Online documentation: tartine-studio.dev/assets/drop-in-water/docs/

Please include your Unity and URP versions, the target platform, a screenshot of the Inspector and any Console errors.

Tools > Tartine Studio > Drop-In Water > Contact Support opens an email with the version filled in.

17. Version history

- **1.0.0:** first release.