Setup Guide

Welcome to the NWH Dynamic Water Physics 2 setup guide. This section covers everything you need to get started with water physics simulation in Unity.

Getting Started

- Quick Start (QuickStart.html) - Set up your first water object quickly
- Demos (Demos.html) - Explore the included demo scenes

Water System

- Water Object (WaterObject.html) - Main water physics component
- Water Object Manager (WaterObjectManager.html) - Managing multiple water objects
- Supported Water Assets (WaterAssets.html) - Overview of supported water assets

Water Effects

- Water Particle System (WaterParticleSystem.html) - Water particle effects and configuration

Ship Controllers

- Ship Controller (ShipController.html) - Drivable boats
- Advanced Ship Controller (AdvancedShipController.html) - Advanced controller features
- Sail Controller (SailController.html) - Adds sailing functionality to boats
- Submarine (Submarine.html) - Submarine controller and depth management

Ship Components

- Engine (Engine.html) - Engine configuration and propulsion
- Thruster (Thruster.html) - Thruster setup and management
- Rudder (Rudder.html) - Rudder configuration and steering
- Anchor (Anchor.html) - Physics-based anchors

Input

- Input Overview (Input.html) - User input configuration and bindings
- Input System Provider (InputSystemProvider.html) - InputSystem provider setup
- Input States (InputStates.html) - Input state management

Physics & Mass

- Mass From Volume (MassFromVolume.html) - Automatic mass calculation from volume
- Mass From Children (MassFromChildren.html) - Mass calculation from child objects

Multiplayer

- Multiplayer Overview (Multiplayer.html) - Multiplayer integration guide
    - Mirror (Mirror.html) - Mirror networking setup

    - Photon Unity Networking 2 (PhotonUnityNetworking2.html) - PUN2 setup

Resources

- Upgrade Notes (UpgradeNotes.html) - Information for upgrading from previous versions
- Changelog (Changelog.html) - Version history and changes
- Support (Support.html) - Get help and support

Quick Start

This guide explains how to set up a primitive sphere to interact with water. An empty scene will be used for the example.

Sample Prerequisites

Before starting with NWH Dynamic Water Physics 2, import the required samples:

> ⚠ **IMPORTANT**
> **NWH Common Package > Common Base Sample** must be imported first. This sample is required for all NWH package samples to function properly.

Samples are imported via **Package Manager > In Project > NWH Dynamic Water Physics 2 > Samples**.

> ℹ **NOTE**
> Samples are found in the Package Manager under the **"In Project"** category in the left sidebar, NOT under "My Assets" or other categories.

The **Base Sample** contains the main demo scene and is required before importing other Dynamic Water Physics samples (Multiplayer, water integration samples).

Water Object

Warning: If making an open-world game with a large-scale map (> ~4000 units), the floating origin is required (not only for this asset, but in general with game engines that use floating point precision). See floating origin explanation (https://manuel-rauber.com/2022/04/06/floating-origin-in-unity/).

Any physics object that is active and has `WaterObject` attached will interact with water. There are two requirements for `WaterObject` to work: a `Rigidbody` and a `MeshFilter`:

- `MeshFilter` is required so that the `WaterObject` knows which mesh to use for simulation.
- `Rigidbody` does not have to be attached to the same object as `WaterObject`, but it must be present in one of its parents. This allows for composite objects; one Rigidbody with multiple hulls - such as a trimaran.

Example Manual Setup

1. Add a *3D Object* > *Sphere* to the scene.
2. Add a `Sphere Collider` to the Sphere if not automatically added.
3. Add a `Rigidbody` to the Sphere and set its mass to 300. There is also a script called `MassFromMaterial` which can calculate and set the `Rigidbody` mass based on material density and mesh volume, but it is a helper script and not required.
4. Add `WaterObject` to the Sphere. Since the sphere by default has 768 triangles the `Simplify Mesh` option should be used. This option automatically decimates the mesh to a `Target Triangle Count`. A good triangle count is 30 or less for simple objects and around 60 for ship hulls. Using higher triangle count will have a linear performance penalty with minimal quality gain. For the example sphere, 36 is sufficient.

Example Auto Setup

1. Add a *3D Object* > *Sphere* to the scene.
2. Attach `WaterObjectWizard` to the sphere and press *Auto-Setup*.

Center Of Mass

`Center of Mass` is one of the most important settings for a ship. A ship with too high center of mass will **capsize**. `Center of Mass` can be adjusted by unticking the automatic center of mass on the `Rigidbody` and adjusting the values there. Lower `Y` value will make the ship more resistant to capsizing, but setting it below the keel will make the ship lean into the corner instead of away from it.

Water Data Provider

`WaterDataProvider` is a script that tells `WaterObject` where the water is. It is an interface between water systems/assets and DWP2. All flat water assets/shaders use the same `WaterDataProvider` : `FlatWaterDataProvider` , while for wavy assets such as Crest, an asset-specific `WaterDataProvider` has to be used, e.g. `CrestWaterDataProvider` . As of version v2.5, multiple water surfaces can be used in the same scene. This is done by attaching a `Collider` with `isTrigger = true` to the `WaterDataProvider` . As long as the object is inside the trigger, it will use data from that `WaterDataProvider` .

Minimal Setup

1. Add a Cube (or any other mesh) to the scene.
2. Attach `WaterObject` to the Cube. Make sure that a Rigidbody has been added.
3. Press Play. The object will now float at default water height (set under `WaterObject` settings).

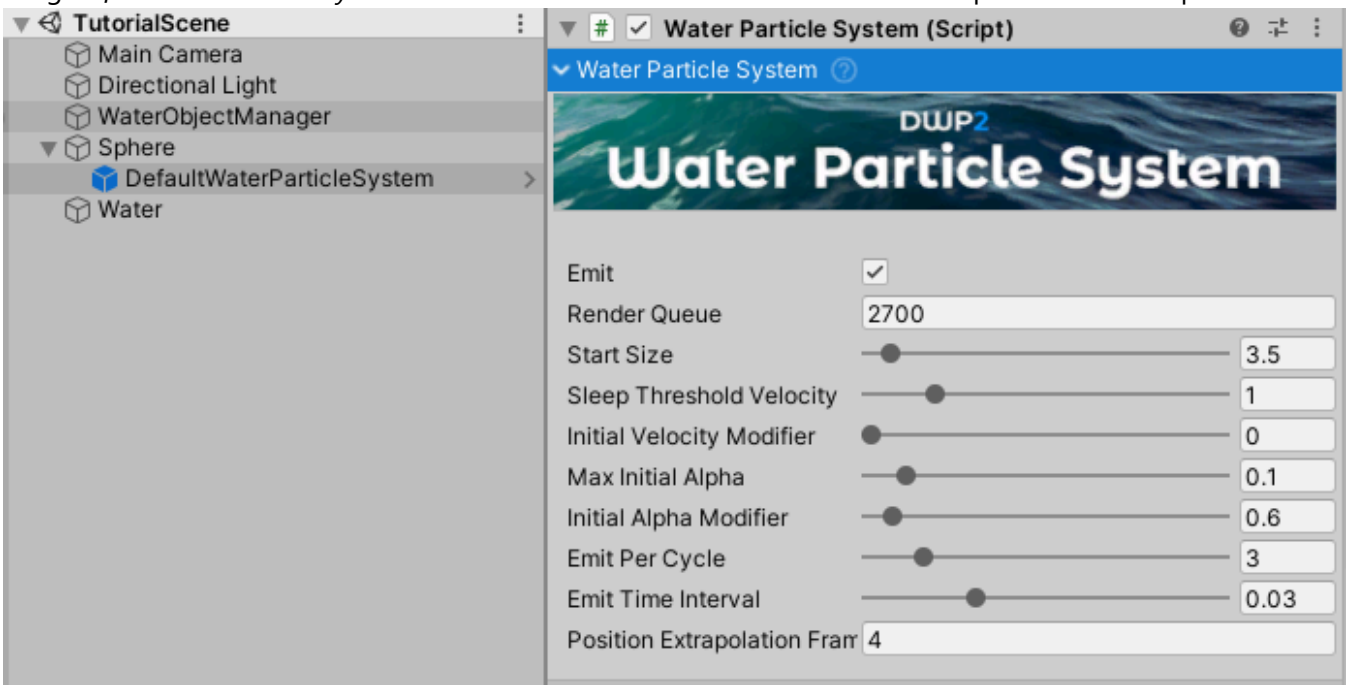
Adding Water

- `FlatWaterDataProvider` can be used to make water height follow a flat primitive plane.
- For more info and 3rd party asset integration check the Water Assets page (WaterAssets.html).

Water Particle System

`WaterParticleSystem` can be used to generate foam. It works with any flat water.

1. Drag *DefaultWaterParticleSystem* from *DWP2 => Resources* into the scene and parent it to the Sphere.



2. Move the Sphere above the water and press play. The sphere falling into the water will generate foam around it based on simulation data. `WaterParticleSystem` and `ParticleSystem` values can be tweaked to suit the needs of the project.

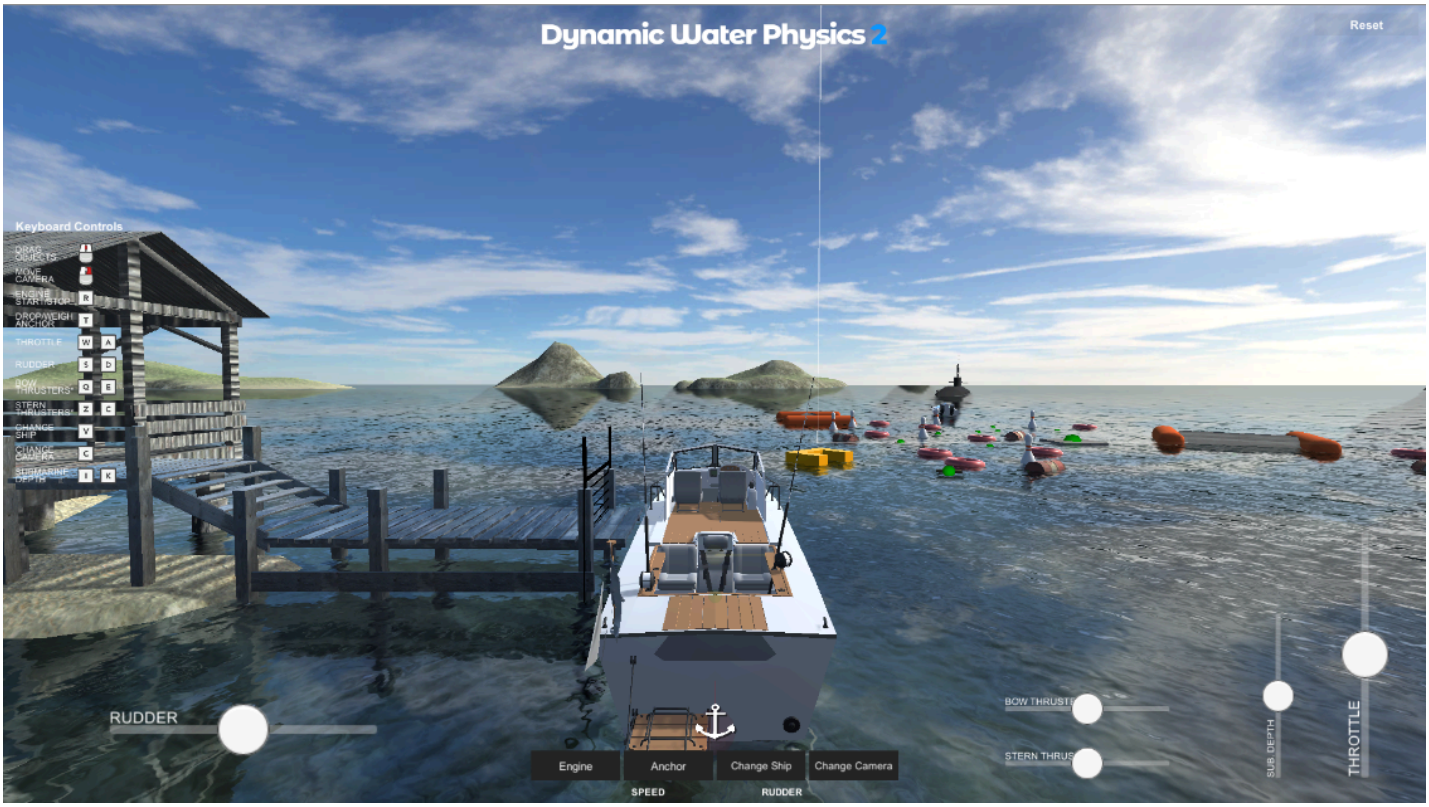
Water Object Wizard

`WaterObjectWizard` is a helper script that sets up a `WaterObject` automatically. It is still recommended to have knowledge of manual setup and how things work, but this script can automate and speed up the setup process. A primitive Sphere will be used, same as in the manual setup section above.

1. Add a *3D Object* > *Sphere* to the scene.
2. Add `WaterObjectWizard` to the newly created Sphere.
3. Tick *Add Water Particle System* (optional). This option is self-explanatory.
4. Click `Auto-Setup` and press Play after the setup is done. The Sphere now floats and generates foam. Next step would be to manually check and tweak the default values, such as `Target Triangle Count`, center of mass, etc.

Demo Builds

Main Demo

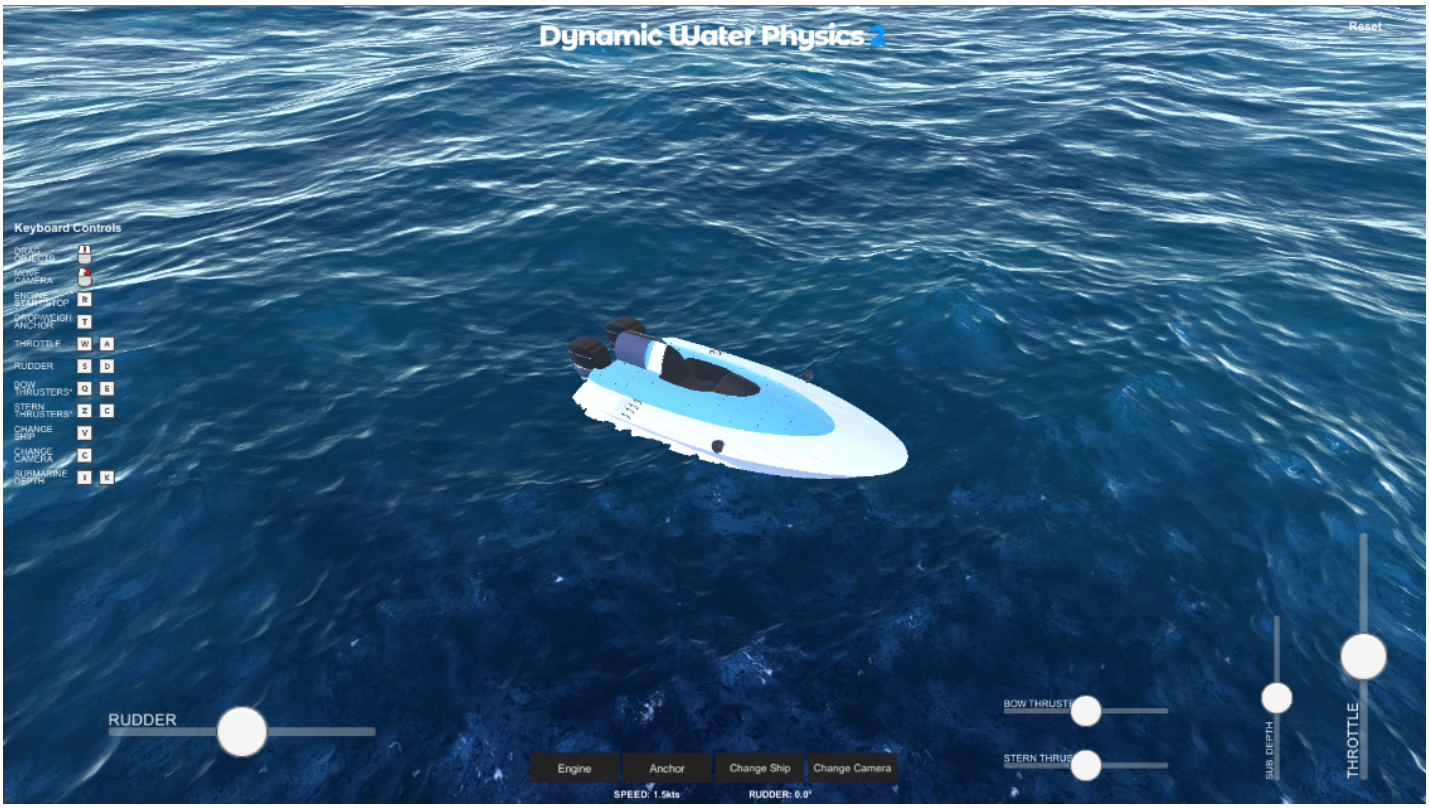


Download

- Windows (https://drive.google.com/file/d/113uDCgYMwfoztdaZpmgwZJUDdzzDGVz3/view?usp=share_link)
- MacOS (https://drive.google.com/file/d/1LfpAxI2pFsEAtEldqsKFX0j0NsglLZAv/view?usp=share_link)
- Linux (https://drive.google.com/file/d/14La08D2xc9v507JKQWLSSAT0o06lrC-s/view?usp=share_link)

Crest Demo

> **Note:** Crest is a 3rd party open source ocean system and requires separate download from the Crest ocean system repository (https://github.com/crest-ocean/crest) (free). URP and HDRP versions of Crest are also available on the Unity Asset Store.

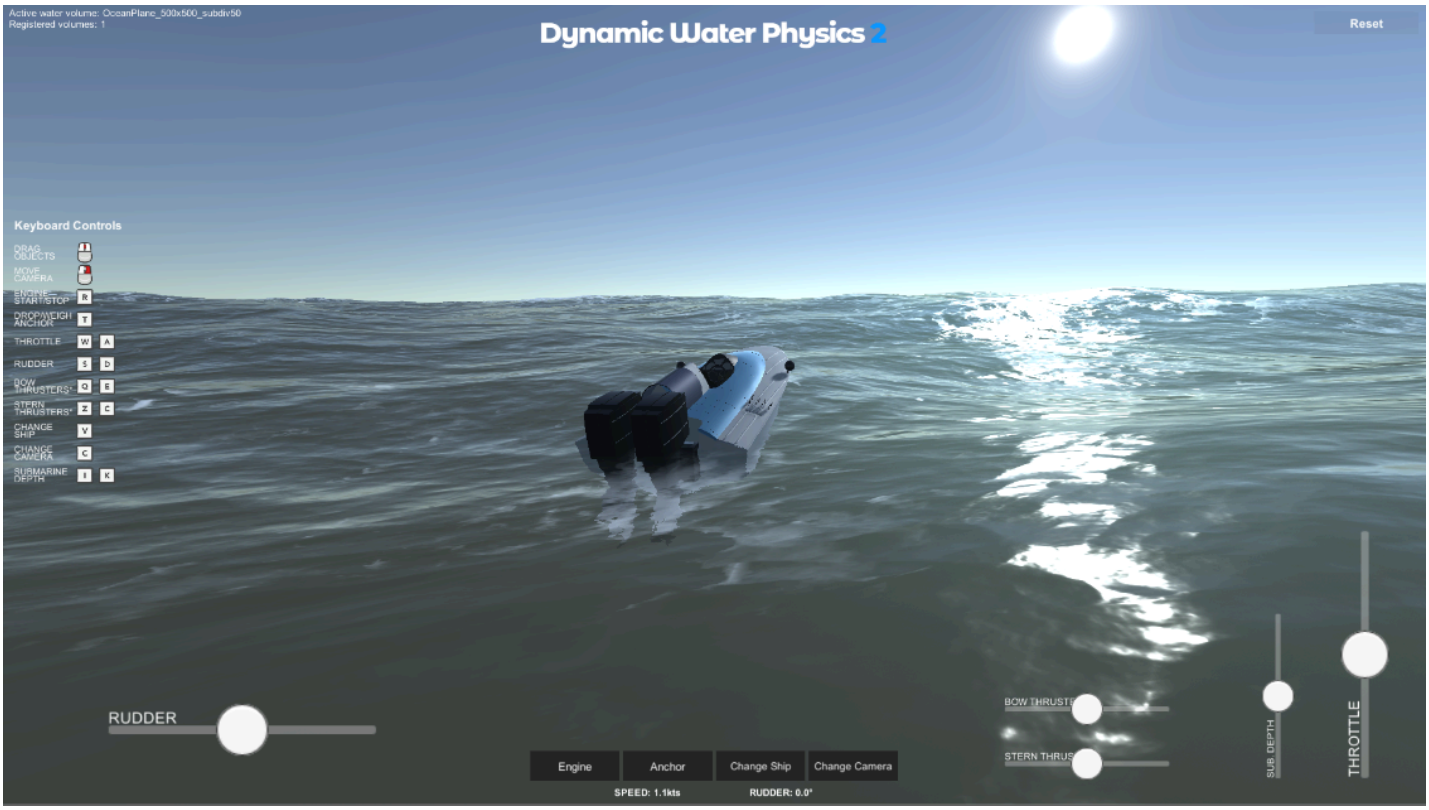


Download

- Windows (https://drive.google.com/file/d/1QhcRyFjRg89uo_bwhn_hZCWe2duwV81c/view?usp=sharing)
- MacOS (https://drive.google.com/file/d/1OQnCBOLCvfYRO5jBIv0B6yTk3v0CFm3F/view?usp=sharing)

Lux Water Demo

> **Note:** Lux Water is a 3rd party ocean renderer and requires separate download from the Lux Water asset on the Unity Asset Store (https://assetstore.unity.com/packages/vfx/shaders/lux-water-119244).



Download

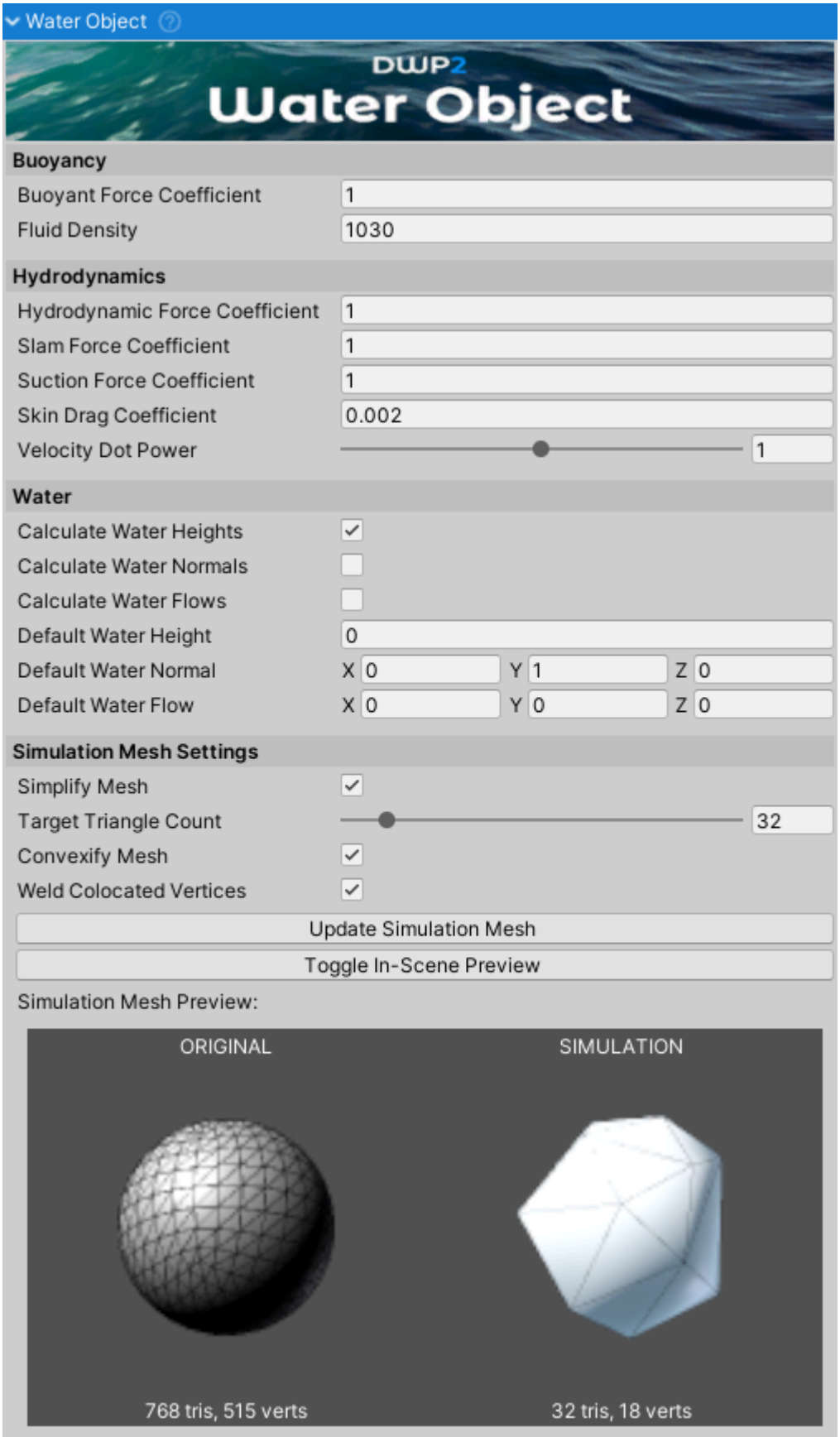
- Windows (https://drive.google.com/file/d/1RtyQf5AXCpZ0ohUkZBT1EnE-4O2pSNaJ/view?usp=sharing)

Multiplayer Demos

Simple demo scene for testing out multiplayer functionality.

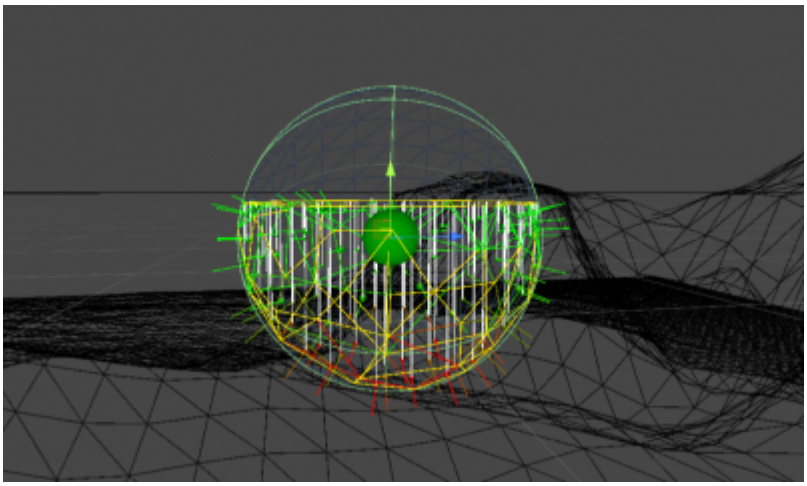
- Mirror (Win64) (https://drive.google.com/file/d/1BN6fn0NoMhC2QpPq3DauVkFrOT4xqRbv/view?usp=sharing)
- Photon Unity Networking 2 / PUN2 (Win64) (https://drive.google.com/file/d/1cquhDkXt-V6I9QqTDWOgxxr1cD3XAaIl/view?usp=sharing)

Water Object



`WaterObject` is the main script of DWP2. It handles all the buoyancy and hydrodynamics calculations.

`WaterObject` gets the data needed for simulation from the *Simulation Mesh*. This is a static mesh that will be used for simulating water/object interaction and can either be the original mesh or a simplified version of it. `WaterObject` provides basic tools for mesh triangle decimation, removal of co-located vertices, and convexification - if needed.



Tips

- When using hollow boat/ship hulls it is recommended to use the `Convex` option when generating the simplified mesh.
- `WaterObject` density should be above ~15kg/m. Large volume, extremely low mass objects can cause the object to jump/fly away when falling into the water. Use `MassFromVolume` script to calculate mass from object volume / density.

Field Explanations

Force Coefficients

- `buoyantForceCoefficient` - Multiplier for buoyancy forces. Increase to make the object float higher, decrease to make it sink deeper.
- `hydrodynamicForceCoefficient` - Multiplier for all hydrodynamic forces (pressure drag). Affects how much the water resists object movement.
- `skinDragCoefficient` - Multiplier for friction drag as water flows over the surface. Higher values increase surface drag.
- `slamForceCoefficient` - Multiplier for impact forces when faces enter the water. Higher values create more splash and resistance.
- `suctionForceCoefficient` - Multiplier for forces when faces exit the water. Higher values increase the suction effect.

Simulation Settings

- `fluidDensity` - Density of the fluid in kg/m³. Default is 1030 (seawater). Use 1000 for fresh water.
- `targetTriangleCount` - Number of triangles in the simplified simulation mesh. Lower values improve performance but reduce accuracy. Range: 8-256.
- `velocityDotPower` - Exponent applied to the velocity-normal dot product. Higher values reduce hydrodynamic forces when triangles are nearly parallel to water flow.

Mesh Processing

- `simplifyMesh` - Reduces triangle count of the simulation mesh to targetTriangleCount for better performance.
- `convexifyMesh` - Makes the simulation mesh convex. Useful for hollow hulls or partially open meshes.
- `weldColocatedVertices` - Merges vertices at the same position. Improves performance and is recommended.

Public Methods

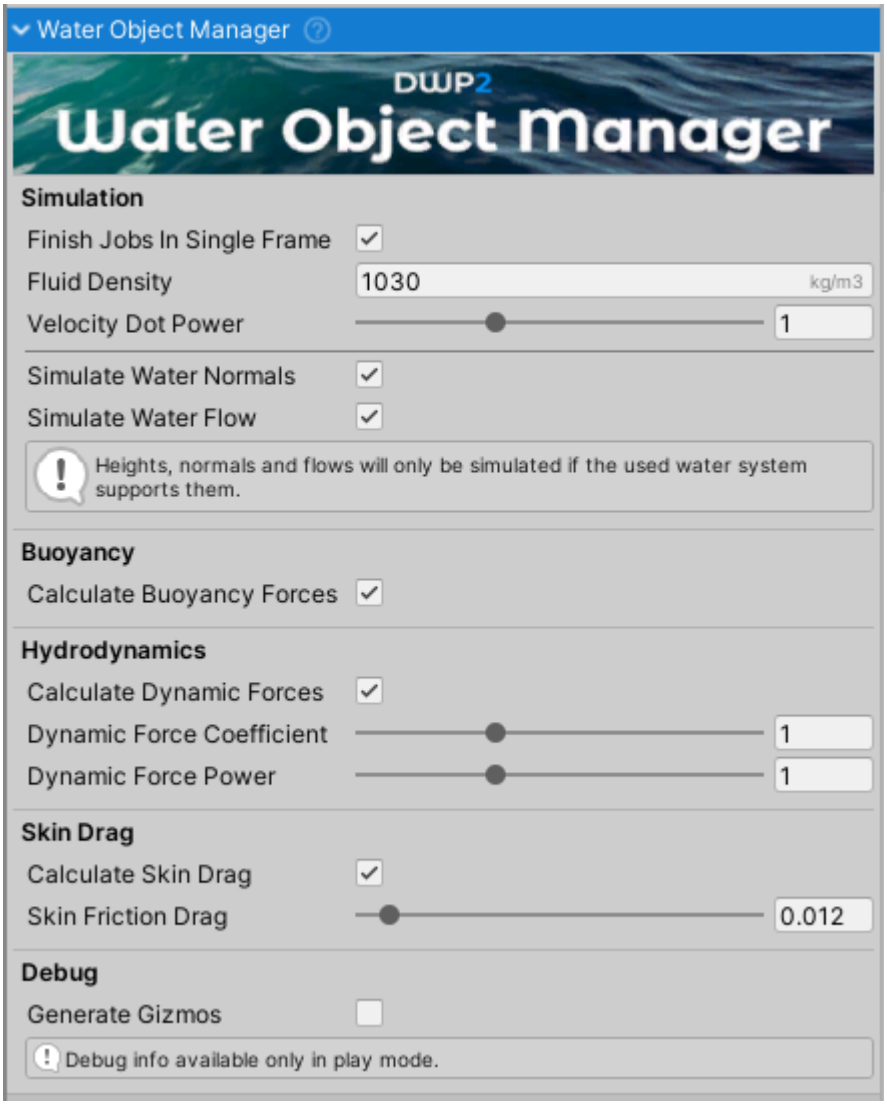
- `GetWaterHeightSingle(Vector3 point)` - Returns water height at a single point in world space. Uses the current WaterDataProvider or falls back to defaultWaterHeight if none is active.
- `IsTouchingWater()` - Returns true if any part of the object is touching water (at least one triangle is submerged). Cache the result if calling this frequently to avoid iterating through all triangle states each frame.
- `StartSimMeshPreview()` - Enables visualization of the simulation mesh in the scene view by temporarily replacing the MeshFilter's mesh. Useful for inspecting mesh simplification results.
- `StopSimMeshPreview()` - Disables simulation mesh preview and restores the original mesh to the MeshFilter.
- `GenerateSimMesh()` - Generates or regenerates the simulation mesh based on current settings (simplifyMesh, convexifyMesh, weldColocatedVertices, targetTriangleCount). Call this after changing any mesh processing settings.

Instantiating at Run-time

`WaterObject` s can be instantiated at run-time like any other GameObject.

Water Object Manager

> **Note:** `WaterObjectManager` has been removed in v2.5. This page exists for backwards compatibility purposes.



`WaterObjectManager` is the main script of Dynamic Water Physics 2 and must be present in the scene for water/object interaction to work. It does not matter to which object it is attached, it just needs to be present in the scene. `WaterObjectManager` fetches the data from all the WaterObjects (WaterObject.html) in the scene, processes it, and sends it to a job which performs physics calculations using multiple CPU cores.

Fields

Simulation Settings

- `Finish Jobs In Single Frame` - If enabled, jobs will be finished on the same frame they started. Disabling improves performance but adds a one-frame delay to water/object interaction. This delay is negligible in most cases. Default is disabled.
- `Fluid Density` - Density of the simulated fluid in kg/m^3. Default is 1030 (salt water).
- `Velocity Dot Power` - Controls the velocity/force relationship. A value of 1 should be used for normal water. Values below 1 reduce force response, values above 1 increase it.

Force Calculation

- `Calculate Buoyancy Forces` - When enabled, buoyancy forces will be calculated.
- `Calculate Dynamic Forces` - When enabled, hydrodynamic forces will be approximated. Disable to use only buoyancy.
- `Dynamic Force Coefficient` - Multiplier for the calculated hydrodynamic forces.
- `Dynamic Force Power` - Exponent applied to the calculated hydrodynamic force.
- `Calculate Skin Drag` - When enabled, friction drag is calculated from fluid passing over surfaces.
- `Skin Friction Drag` - Skin friction coefficient. Use 0 or near 0 for water; higher values for viscous fluids.

Water Interaction

- `Simulate Water Normals` - When enabled, water surface normals are used when calculating forces. Has no effect if the water system does not support normals.
- `Simulate Water Flow` - When enabled, water flow is used when calculating forces. Has no effect if the water system does not support flow.

Debugging

- `Generate Gizmos` - When enabled, gizmos are generated from simulation data for visualization in the scene view.

Instantiating WaterObjects at Runtime

> **Warning:** When a `WaterObject` is added to the scene during play mode, it will not be automatically registered. This requires re-allocating memory for jobs, which is expensive and should ideally be done during loading screens.

There are two approaches for adding objects to the scene during runtime:

Method 1: Immediate Synchronization

Use this method when instantiating a low number of triangles:

- Instantiate the object
- Immediately call `WaterObjectManager.Instance.Synchronize()`

This ensures the object is immediately registered and simulated.

Method 2: Pre-instantiate and Deactivate

Use this method when you need to add many objects:

- Instantiate all objects during scene load
- Keep them deactivated
- `WaterObjectManager` will allocate memory for inactive objects without simulating them
- When needed, simply activate the object (no need to call `Synchronize()`)

See Also

- WaterObject (WaterObject.html) - Individual water physics component for each object
- WaterAssets (WaterAssets.html) - Supported water systems for interaction
- Quick Start (QuickStart.html) - Getting started with water physics simulation

Supported Water Assets

`WaterDataProvider` scripts are interfaces between the 3rd party water assets and Dynamic Water Physics 2. They tell the WaterObject where the water is.

Using Multiple Assets / Water Types at Once

Multiple water types can be used in the same scene at the same time. This is achieved through triggers (`Collider`s with `isTrigger` set to true) attached to the same `GameObject` as the `WaterDataProvider` in question.
By default these colliders are created automatically on `Awake` and are set to cover the whole world. However, it is possible to use a `WaterDataProvider` just for a small part of the scene - such as a lake.

- Attach a `Collider` of any type (e.g. `SphereCollider`) to the `GameObject` containing `WaterDataProvider`.
- Tick `Is Trigger` on the collider.
- Adjust the size/radius of the collider to cover the area you want the `WaterDataProvider` to have the effect on.

If there are multiple trigger volumes they will act as a queue, meaning the one that the object last entered will be currently active.

Flat Water Integration

`FlatWaterDataProvider` can be used for all flat water systems.
It can even be used with wavy water systems if the waves have 0 amplitude to improve performance - sometimes drastically as the water heights are always queried with wavy water system, even if there are no waves.

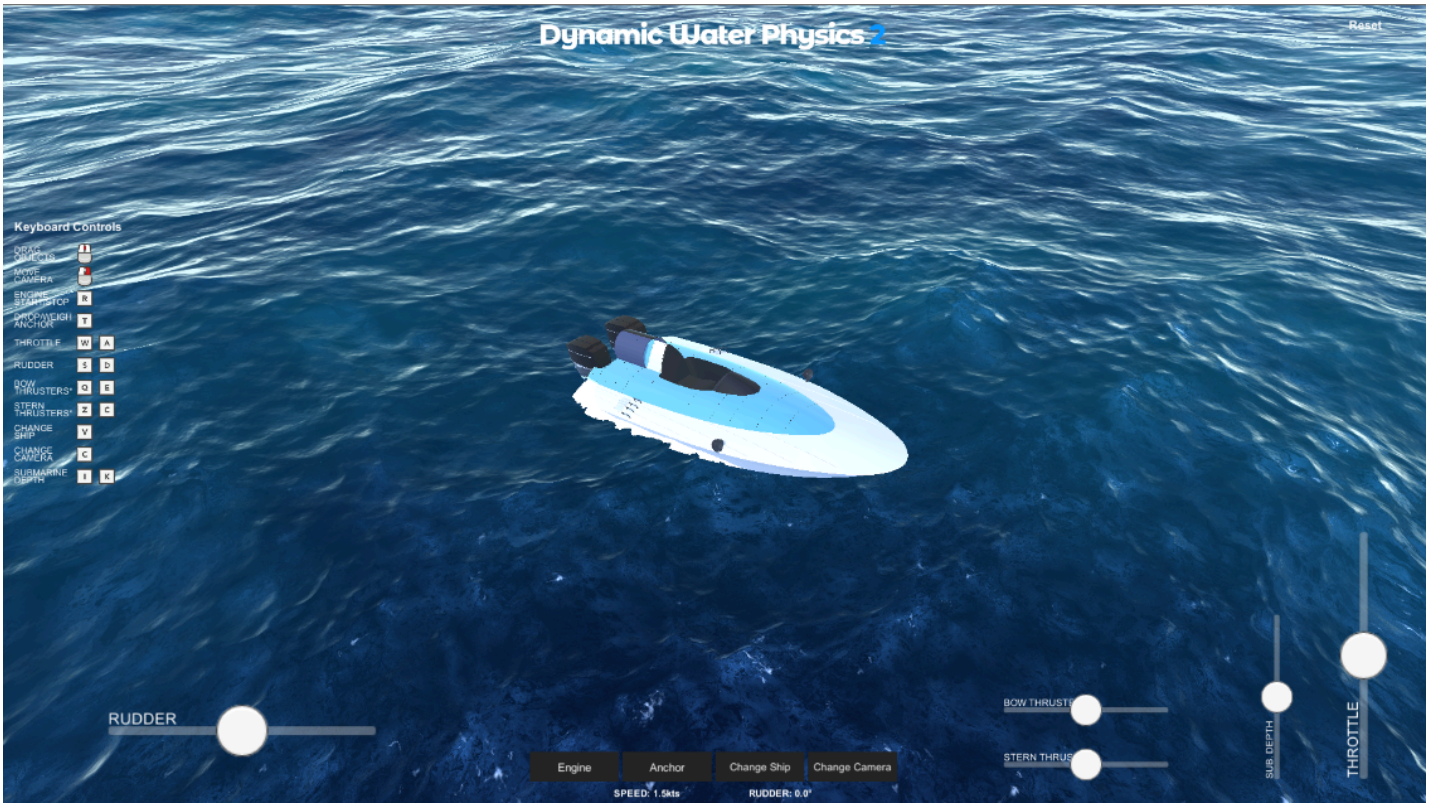
1. Attach `FlatWaterDataProvider` to the `GameObject` representing the water.
2. `WaterObject`s will now float at the water `transform.y` position.

Unity HDRP Water

Unity HDRP water requires HDRP rendering pipeline and will not work with URP.

1. Import the `Unity HDRP Water Integration` sample from the Package Manager.
2. Attach the `UnityHDRPWaterDataProvider` to the scene `GameObject` containing the HDRP water script.

Crest



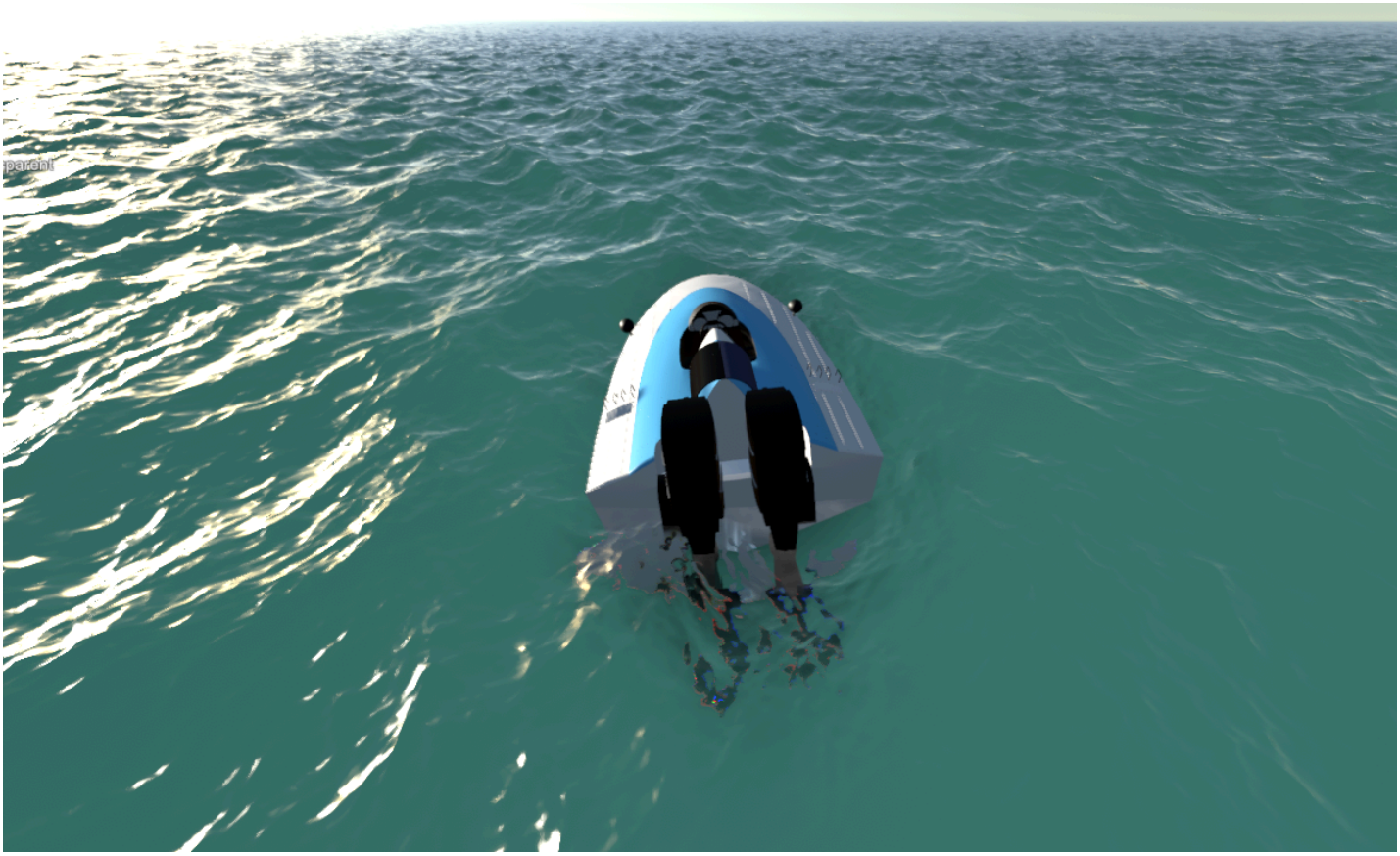
Dynamic Water Physics 2 is compatible with Crest v10 or newer. Older versions have different API. Crest supports water heights, normals and flows.

1. Import the `Crest Integration` sample from the Package Manager.
2. Find in the scene the `GameObject` containing the `CrestWaterRenderer` and attach `CrestWaterDataProvider` to it.

Stylized Water 3

1. Import the `Stylized Water 3 Integration` sample from the Package Manager.
2. Attach the `StylizedWater3WaterDataProvider` to the `GameObject` containing the water script.
3. Assign the wave profile.

KWS



1. Import the `KWS Integration` sample from the Package Manager.
2. Attach `KWS Water Data Provider` to the GameObject containing `Water System` (this game object is named *Water* in KWS demos).

R.A.M. (River Auto Material)

`RAMWaterDataProvider` supports water heights, normals and flow and it inherits from `RaycastWaterDataProvider` .

1. Import the `RAM Integration` sample from the Package Manager.

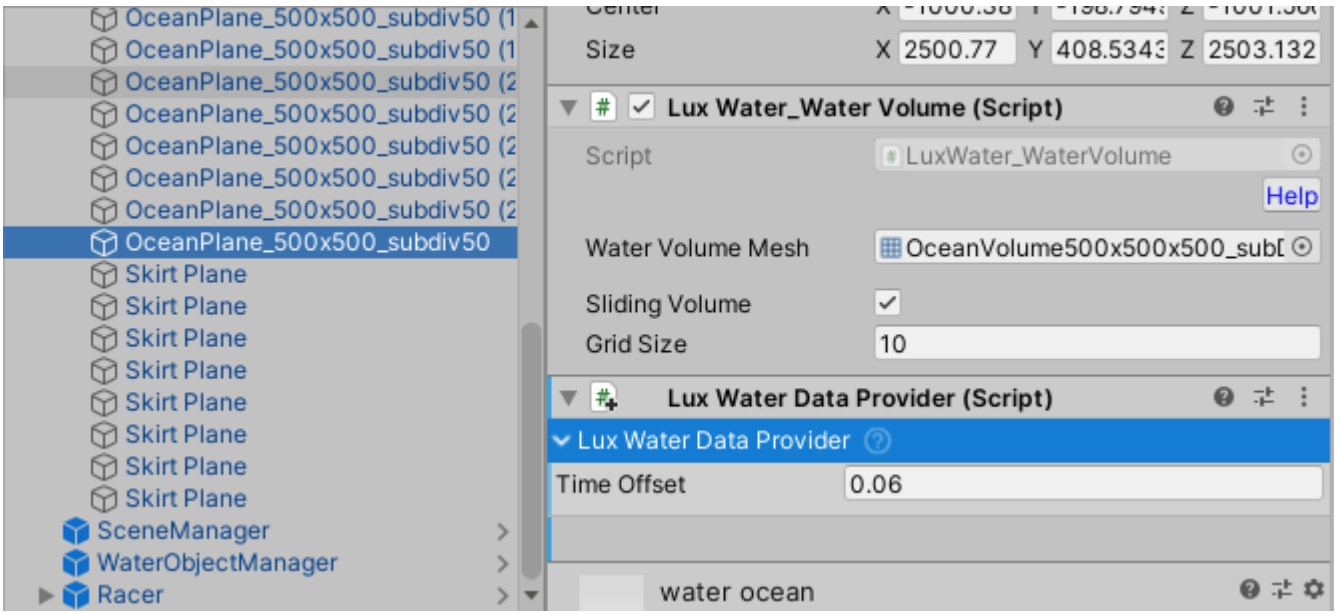
2. Set up the scene as if using flat water, minus FlatWaterDataProvider.
3. Add RAMWaterDataProvider to the scene. It does not have to be attached to any specific object.
4. Make sure that the RAM object has a MeshCollider attached. This is required for Raycasts to work.
5. Assign Water Layer to the River Auto Material water. Assign Object Layer to all the WaterObjects in the scene. This is an important step as the script will disable physical collisions between the two layers to prevent the WaterObjects from sitting on top of the mesh collider that R.A.M. uses instead of interacting with water.
6. R.A.M. setup is ready to go.

Stylized Water 2

Unlike other `WaterDataProviders`, the one for Stylized Water 2 is included with the Stylized Water 2 asset instead of DWP2.

1. Set up the scene as per quick start guide for flat water, minus FlatWaterDataProvider.
2. Go to Help -> Stylized Water 2, and click the "Install integration" button. Wait until scripts have finished compiling.
3. Add `StylizedWaterDataProvider` to the object containing OceanRenderer script.

LUX



1. Import the `Lux Water Integration` sample from the Package Manager.
2. Attach `LuxWaterDataProvider` to `LuxWater_WaterVolume`.

Ceto

- Support removed in v14

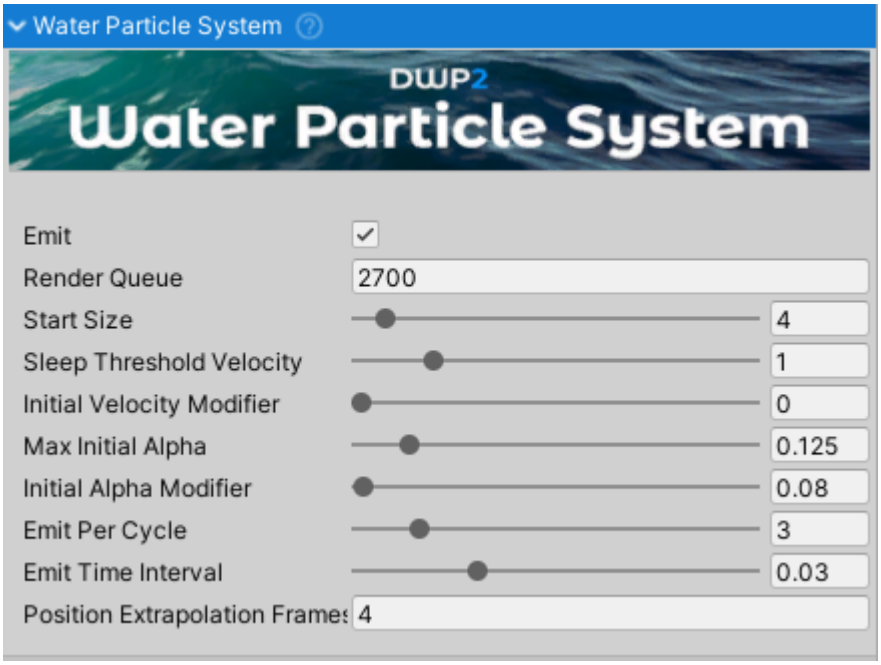
Ocean Next Gen

- Support removed in v14

SUIMONO

- Support removed in v14

Water Particle System



Water Particle System is a component that generates particles based on simulation data from Water Object Manager. It can be added to any Water Object.
Water Particle System has been rewritten from the ground up for DWP2 and now it has virtually no performance or memory overhead except for the cost of the Unity's Particle System it uses to render particles.

> **Warning:** Water Particle Systems emits only along X-Z axis and does not work with wavy water assets. For that asset-specific foam has to be used (if available).

Fields

- `Emit` - Particles will only be generated when this field is ticked.
- `Render Queue` - Render queue of the particle material. If particles are rendered behind the water increase the value to be just above the value of the water's render queue.
- `Surface Elevation` - Height above water surface at which the particles will be emitted.
- `Start Size` - Starting diameter of the particle.
- `Sleep Threshold Velocity` - If rigidbody's velocity is below this value particles will not be emitted. Do not set to 0 as that will result in (invisible) particles constantly being generated, even when object is still.
- `Initial Velocity Modifier` - Velocity at the point of contact with water is multiplied by this value to get the initial particle velocity. If set too high it will seem as if the particles are flying away from the object.
- `Max Initial Alpha` - Maximum initial alpha (transparency) of the foam. If set to 1 foam will be opaque, 0 and it will be invisible.
- `Initial Alpha Modifier` - Higher contact force with water will result in higher initial alpha (up to Max Initial Alpha). This field sets the sensitivity of alpha related to the force.
- `Emit Per Cycle` - How many particles should be emitted in each cycle? If there are not enough contact points with water less particles may be emitted.
- `Emit Time Interval` - Interval between emission cycles in seconds.
- `Position Extrapolation Frames` - To counteract the initial fade-in and apparent lag of the particles, the emission position is predicted a number of frames in advance. If this number is set too high particles will appear as if emitting in front of the object.

Ship Controller

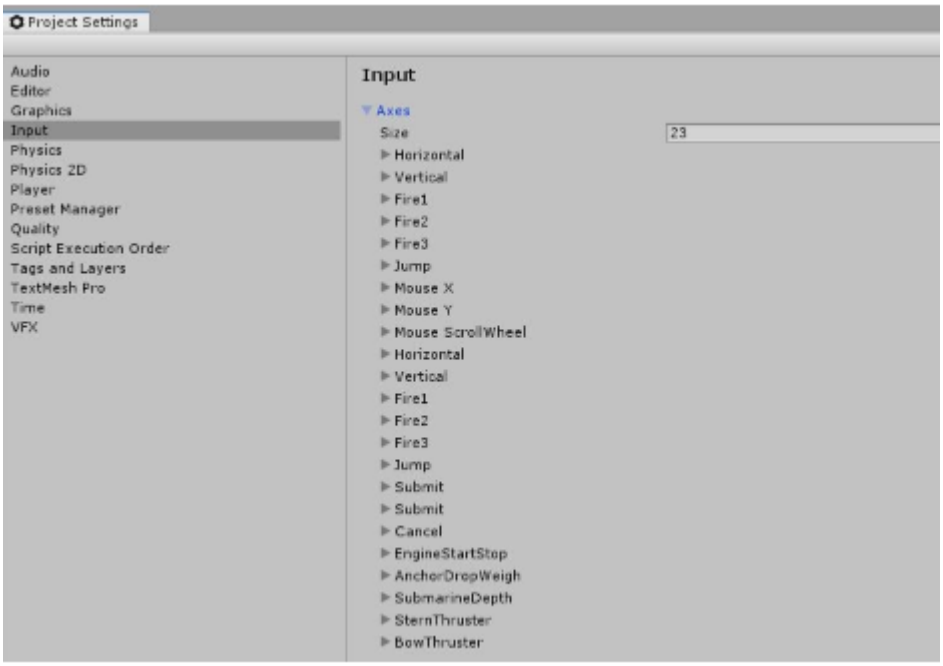
Step-by-step Setup Guide for Ship Controller

Input

If you are using the legacy InputManager, a few input bindings need to be set up for everything to work properly. These can be added under Edit => Project Settings => Input.
These are:

- `EngineStartStop` (only positive button)
- `AnchorDropWeight` (only positive button)
- `LeftThrottle` (both positive and negative button)
- `RightThrottle` (both positive and negative button)
- `SternThruster` (both positive and negative button)
- `BowThruster` (both positive and negative button)
- `SubmarineDepth` (both positive and negative button)

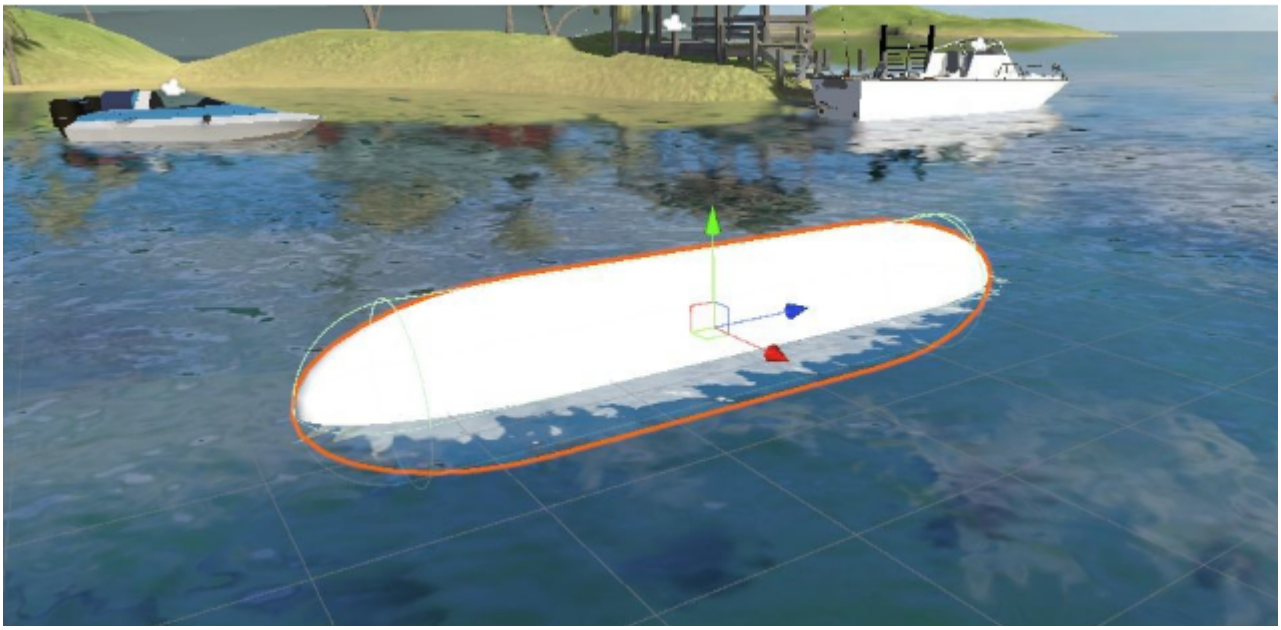
If a function related to the input binding is not needed the binding can be skipped. E.g. if the submarine depth is not needed that input binding can be skipped.



Adding a Ship

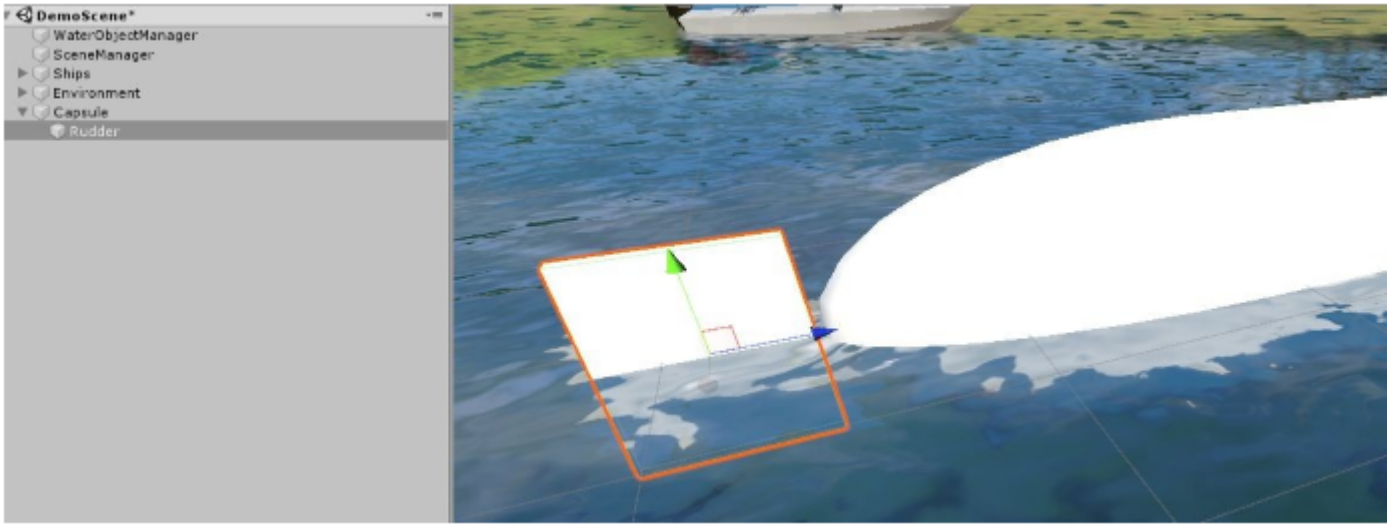
For the example setup a primitive capsule will be used (GameObject menu

> 3D Object > Capsule).



1. Add the ship object into the scene and tag it `Ship` (add a new tag if it does not exist). The tag is only necessary so that the ship changer can find your ship.
2. Add `WaterObjectWizard` component to the ship object. Tick the particle system option and click on Auto-Setup button. Click play - the object will float now. For manual `WaterObject` setup follow the guide here (QuickStart.html).
3. Add `AdvancedShipController` component to the parent object (the one containing the `Rigidbody`).
4. Add `CenterOfMass` component to the parent object and adjust center of mass to be near the bottom of the ship (green sphere). If this is not done the ship will most likely tilt to the side since the center of mass in Unity is calculated as a center of volume of all of the Rigidbody colliders, which is unrealistic for ships which generally have the center of mass near the keel.
5. Add the following Axes under Edit => Project Settings => Input if they do not already exist.
    - `EngineStartStop` (only positive button)
    - `AnchorDropWeight` (only positive button)
    - `SubmarineDepth` (both positive and negative button)
    - `LeftThrottle` (both positive and negative button)
    - `RightThrottle` (both positive and negative button)
    - `SternThruster` (both positive and negative button)
    - `BowThruster` (both positive and negative button)

Rudders

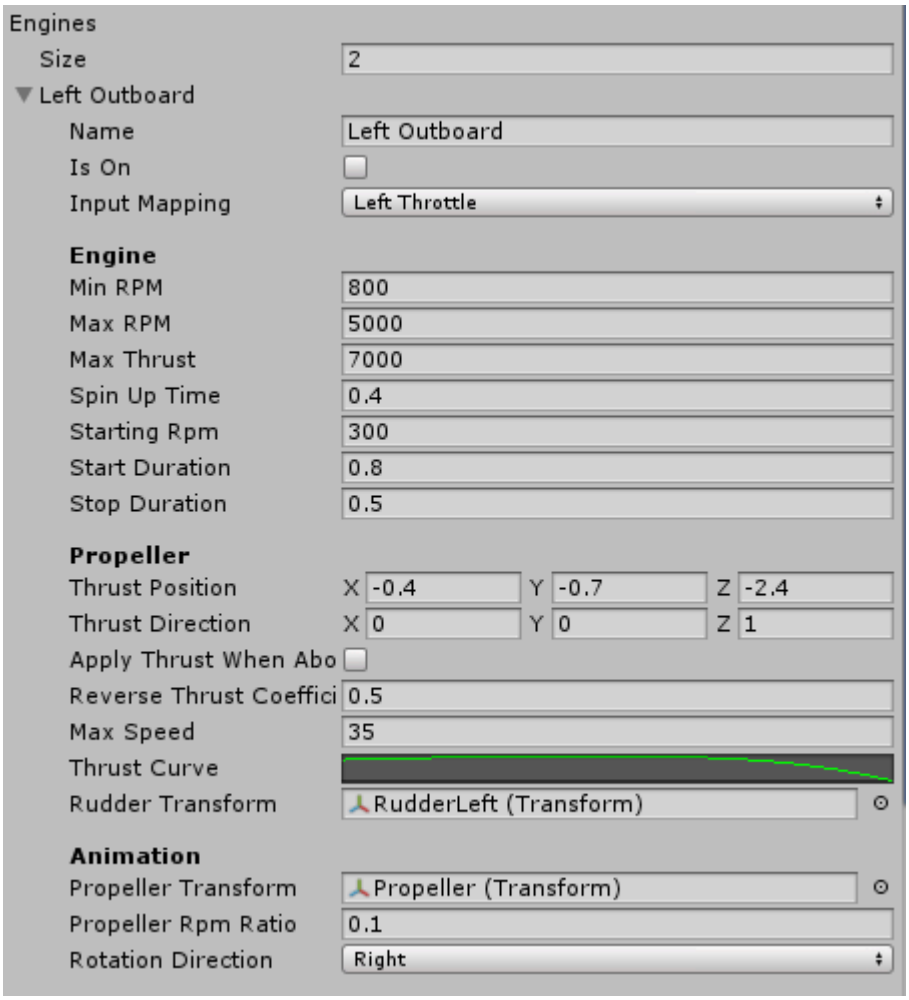


- Add a rudder (in this case just a scaled cube) to the ship.

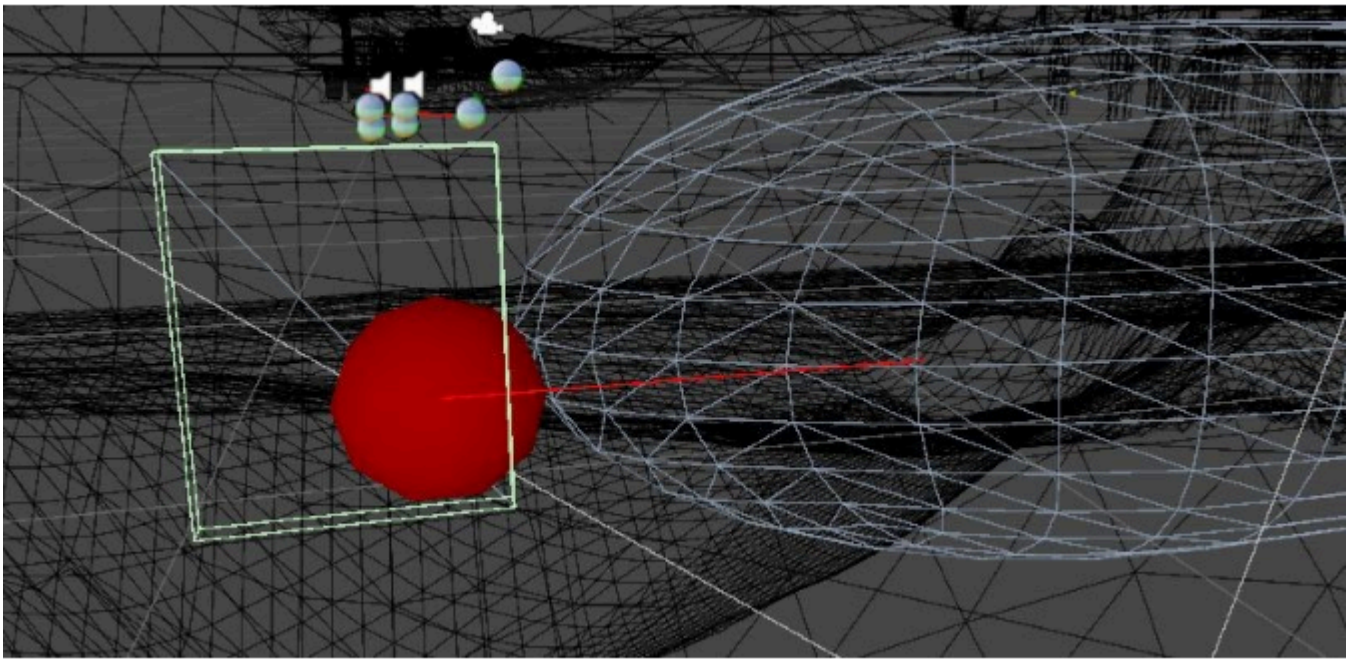
1. Assign the rudder transform (in this example the scaled cube) to the `Rudder Transform` field under *Rudders* foldout.

2. Add `WaterObject` component to the rudder so it too can interact with water.
3. Add a `Camera` of any type to the ship object (as a child) and tag it `ShipCamera`.
4. Press play and cycle to your ship using the `V` button (default change ship button). The boat is now floating and the rudders turn. If the rudders turn around wrong axis the rotation of the model needs to be fixed. Check the Fixing Model Rotation guide (http://nwhvehiclephysics.com/doku.php/Setup/FixingModelRotation) for instructions on how to fix the model rotation.

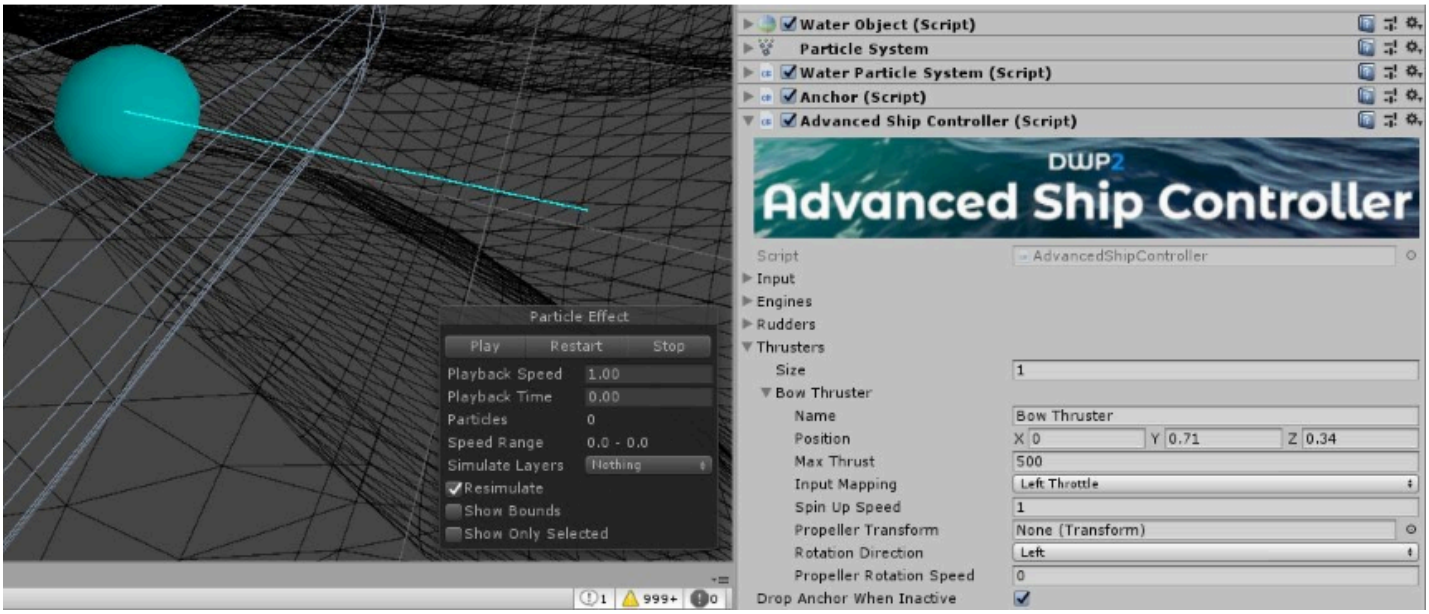
Engines



1. To make ship move an engine and propeller are needed. Ship Controller assumes that there is one propeller per engine. Add an engine under Engines tab and set the wanted values (hover over each value to see what it does). If thrust position is above the water thrust will not be applied (if `Apply Thrust When Above Water` is left unchecked).

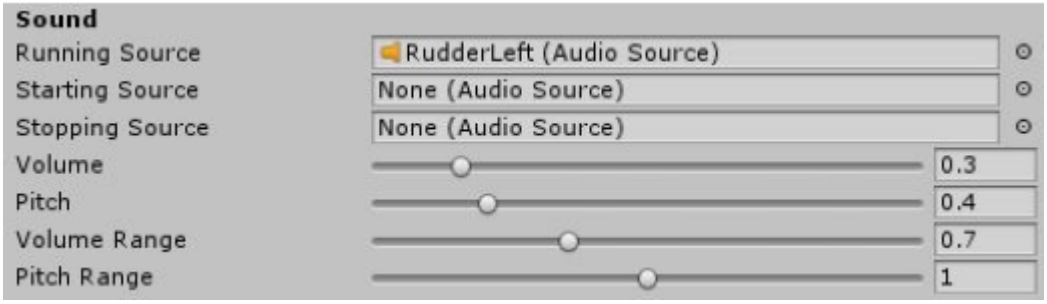


Thrusters



Larger ships usually have bow and/or stern thrusters to help them maneuver. Thruster is indicated as a blue sphere with a line indicating direction of thrust when positive input is pressed.

Sound



Engine sound is achieved through simple pitch modulation. To set up sound:

- Add an `AudioSource` to the ship object and assign a looped engine sound clip to it. Drag the `AudioSource` to the field `Running Source` under Engine foldout.
- Same should be done for `Starting Source` and `Stopping Source` fields. These fields are options. Adjust the `Start Duration` and `Stop Duration` fields to be somewhat shorter than the length of starting and stopping clips.

Advanced Ship Controller

Engine

For information about engine configuration and setup, see Engine (Engine.html).

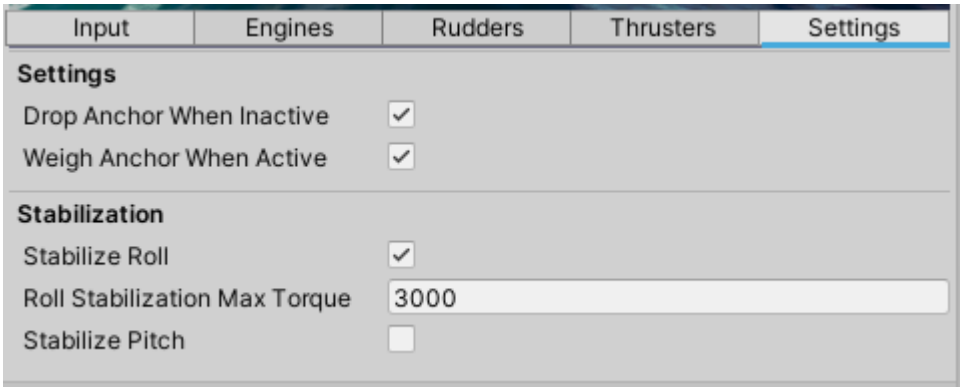
Rudder

For information about rudder configuration and setup, see Rudder (Rudder.html).

Thruster

For information about thruster configuration and setup, see Thruster (Thruster.html).

Settings



These are general ship settings that apply to the overall ship behavior and stability.

Fields

- `Drop Anchor When Inactive` - if there is an `Anchor` component attached to this ship the Anchor will be dropped on sleep.
- `Weigh Anchor When Inactive` - if there is an `Anchor` component attached to this ship the Anchor will be weighed (raised) when ship wakes up.
- `Stabilize Roll` - if true a torque will be applied to the ship to try and negate any roll.
- `Roll Stabilization Max Torque` - maximum torque that will be used for roll stabilization.
- `Stabilize Pitch` - if true a torque will be applied to the ship to try and negate any pitch.
- `Pitch Stabilization Max Torque` - maximum torque that will be used for pitch stabilization.
- `referenceWaterObject` - Reference to the primary WaterObject component used for water height queries. Automatically finds a child WaterObject if not set.

Public Properties and Methods

Properties

- `SpeedKnots` - Current ship speed in knots (1 knot = 1.852 km/h). Calculated from the base Speed property.
- `Anchor` - Reference to the ship's Anchor component. Automatically assigned at startup.

Methods

- `GetPitchAngle()` - Returns the ship's pitch angle in degrees. Positive values indicate bow up, negative values indicate bow down.
- `GetRollAngle()` - Returns the ship's roll angle in degrees. Positive values indicate starboard side down, negative values indicate port side down.

Sail Controller

Quick Start

Calculates lift and drag forces based on wind conditions, sail geometry, and angle of attack.

Prerequisites

Before setting up a sailing ship, ensure you have:

- A ship with `WaterObject` component and `Rigidbody` (follow the Ship Controller guide (ShipController.html) first)
- `AdvancedShipController` component on the ship (for input handling)
- Input bindings set up (see the Input guide (Input.html))

Setting Up Wind Generation

1. Create an empty GameObject in your scene and name it `WindGenerator`.
2. Add the `WindGenerator` component to it.
3. Configure the wind parameters:
    - `Base Direction` - Primary wind direction in degrees (0 = north/+Z axis)
    - `Base Speed` - Average wind speed in m/s (typical: 5-20 m/s)
    - `Max Direction Variation` - Angular deviation during gusts (15-45 degrees)
    - `Max Speed Variation` - Speed deviation from base speed
    - `Min/Max Variation Interval` - Time between wind changes

Note: Only one `WindGenerator` should exist per scene as it uses a singleton pattern.

Creating a Sail Preset

1. Right-click in the Project window and select `Create > NWH > DWP2 > SailPreset`.
2. Name the preset (e.g., "SquareSail" or "LateenSail").
3. Configure the aerodynamic curves:
    - `Lift Coefficient Vs AoA Curve` - How lift varies with angle of attack
    - `Drag Coefficient Vs AoA Curve` - How drag varies with angle of attack
    - `Lift Scale` and `Drag Scale` - Global force multipliers

The preset includes default curves suitable for most sail types. Different sail designs (square, lateen, bermuda) can be represented by adjusting these curves.

Setting Up the Sail

The sail is defined by four corner transforms arranged in a clockwise pattern:

1. Create four empty GameObjects as children of your ship to represent sail corners:
    - `Corner_A` - Bottom front (bottom left for square sails)
    - `Corner_B` - Top front (top left for square sails)
    - `Corner_C` - Top rear (top right for square sails)
    - `Corner_D` - Bottom rear (bottom right for square sails)
2. Position these corners to define your sail shape. For a simple square sail:
    - Place A and D at the bottom of the mast
    - Place B and C at the top of the mast

    - Adjust the spacing to create the desired sail width and height
3. Add the `SailController` component to a child GameObject of your ship (the one with the `Rigidbody`).
4. Assign the corner transforms to the SailController:
    - Drag `Corner_A` to field `a`
    - Drag `Corner_B` to field `b`
    - Drag `Corner_C` to field `c`
    - Drag `Corner_D` to field `d`
5. Assign the `Sail Preset` you created earlier.
6. Adjust `Air Density` if needed (default 1.225 kg/m³ is standard sea level).

Note: Corner transforms can be attached to different parents to enable sail furling or complex rigging. For example, attach top corners to a rotating boom for realistic sail control.

Adding Sail Rotation (Optional)

To allow player-controlled sail rotation:

1. Create a GameObject as a child of your ship (e.g., "SailBoom").
2. Parent the sail corner transforms that should rotate with the boom (typically the rear corners C and D).
3. Add the `SailRotator` component to the boom GameObject.
4. Configure:
    - `Rotation Axis` - Local axis to rotate around (default: Y-axis)
    - `Rotation Speed` - Rotation speed in degrees per second
5. Add a `RotateSail` input axis in Unity's Input Manager:
    - Positive button: D or Right Arrow
    - Negative button: A or Left Arrow

Note: The SailRotator reads input from the parent `AdvancedShipController` component.

Testing the Setup

1. Press play and select your ship using the `V` key.
2. The sail should now generate force based on wind conditions.
3. Use the scene view with Gizmos enabled to see:
    - Sail corners (white spheres)
    - Sail directions (red = right, green = up, blue = forward)
    - True wind (green arrow)
    - Apparent wind (yellow arrow)
    - Sail force (red arrow)
    - Ship velocity (magenta arrow)

How It Works

Apparent Wind Calculation

The sail system calculates **apparent wind** - the wind experienced by the moving vessel. Apparent wind combines the true wind and the ship's velocity, which is why a ship can sail faster than the wind itself and why wind direction changes as the ship moves.

Force Generation

The sail generates two types of forces:

1. **Lift Force** - Acts perpendicular to the sail surface
    - Direction determined by the sail's right vector
    - Magnitude based on lift coefficient curve at current angle of attack
    - Primary propulsion force
2. **Drag Force** - Acts in the direction of apparent wind
    - Always opposes relative motion through air
    - Magnitude based on drag coefficient curve at current angle of attack

Total force depends on air density, apparent wind speed, and sail area.

Angle of Attack

The angle between the sail's forward direction and the apparent wind determines the coefficients:

- **0° to ±45°** - Efficient sailing angles with high lift
- **±45° to ±90°** - Reduced efficiency, transitioning to drag-dominated
- **±90° to ±180°** - Luffing (sail flapping), mostly drag

Heel Compensation

The system compensates for vessel heel (roll) by scaling forces based on sail verticality. This prevents excessive force generation when the sail is nearly horizontal.

Field Explanations

Geometry

- `a, b, c, d` - Four corner transforms defining the sail shape. Must be assigned in clockwise order: front-bottom, front-top, rear-top, rear-bottom. These can be parented to different GameObjects to enable sail furling or complex rigging.

Physics

- `Sail Preset` - ScriptableObject containing lift and drag coefficient curves versus angle of attack. Different sail types (square, lateen, bermuda) require different presets.
- `Air Density` - Air density in kg/m^3 used in force calculations. Standard sea level is 1.225. Can be used as a global force multiplier to tune overall sail power without modifying the preset curves.

Calculated Properties (Runtime)

The following properties are calculated each frame and visible in the inspector during play mode:

Sail Geometry:

- `Sail Center` - Surface-weighted center point where forces are applied
- `Sail Area` - Total sail surface area in m^2
- `Sail Forward` - Forward direction vector of the sail
- `Sail Up` - Up direction vector of the sail
- `Sail Right` - Right direction vector (perpendicular to sail plane)

Wind & Forces:

- `True Wind` - Environmental wind from WindGenerator
- `Ship Velocity` - Current vessel velocity
- `Apparent Wind` - Wind relative to the moving ship (True Wind - Ship Velocity)
- `Angle Of Attack` - Angle between sail forward and apparent wind
- `Sail Force` - Total aerodynamic force vector applied to the ship

Notes and Tips

Multi-Sail Setups

- Add multiple `SailController` components to create ships with multiple sails (main sail, jib, mizzen, etc.)
- Each sail can have its own preset and corner transforms
- Each sail calculates forces independently

Triangular Sails

For triangular sails (like a lateen or bermuda rig):

- Position one corner at the same location as another (e.g., A and D both at the boom)
- The sail area calculation will still work correctly
- Alternatively, use very short edge length between paired corners

Sail Furling

To implement furling:

1. Attach top corners to one parent GameObject
2. Attach bottom corners to another parent GameObject
3. Animate the distance between these parent objects
4. As corners move closer together, sail area automatically reduces
5. Forces scale proportionally with area

Performance

- Sail physics run in `FixedUpdate()` at physics timestep rate
- Gizmo drawing only occurs in editor, not in builds
- Wind generation uses coroutines for smooth transitions

Debugging

Enable Gizmos in the Scene view to visualize:

- Sail shape and corners
- Wind vectors
- Force vectors
- Sail orientation axes

The inspector shows real-time debug info during play mode:

- Current angle of attack
- Force magnitude
- Force vector

Common Issues

Sail not generating force:

- Check that `WindGenerator` exists in scene
- Verify `SailPreset` is assigned
- Ensure all four corner transforms are assigned
- Check that wind speed is non-zero

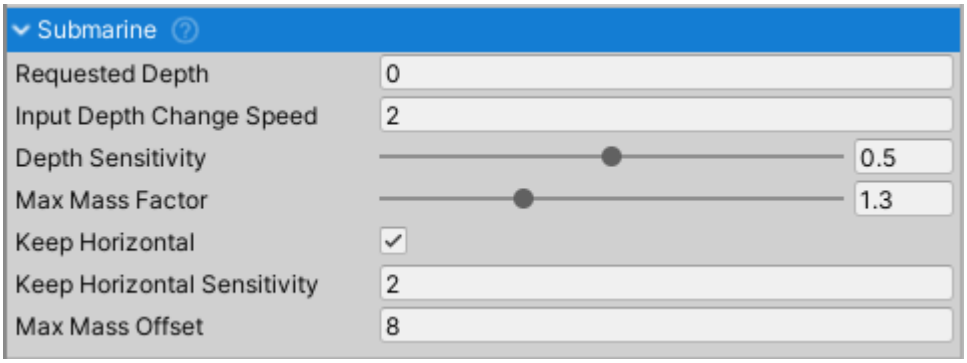
Incorrect force direction:

- Verify corner assignment order (a, b, c, d clockwise)
- Check sail orientation using gizmo arrows
- Ensure `SailForward` points in the expected direction

Sail rotating incorrectly:

- Check `SailRotator` rotation axis direction
- Use negative axis values to reverse rotation
- Verify input binding for `RotateSail` axis

Submarine



A script that can be attached to `AdvancedShipController` to control submarine-specific functions. Works with `VariableCenterOfMass` to adjust buoyancy for depth changes.

> **Note:** VariableCenterOfMass is part of NWH Common (https://nwhcoding.com/Common/). For detailed information, see NWH Common - Variable Center of Mass (https://nwhcoding.com/Common/manual/VariableCenterOfMass.html).

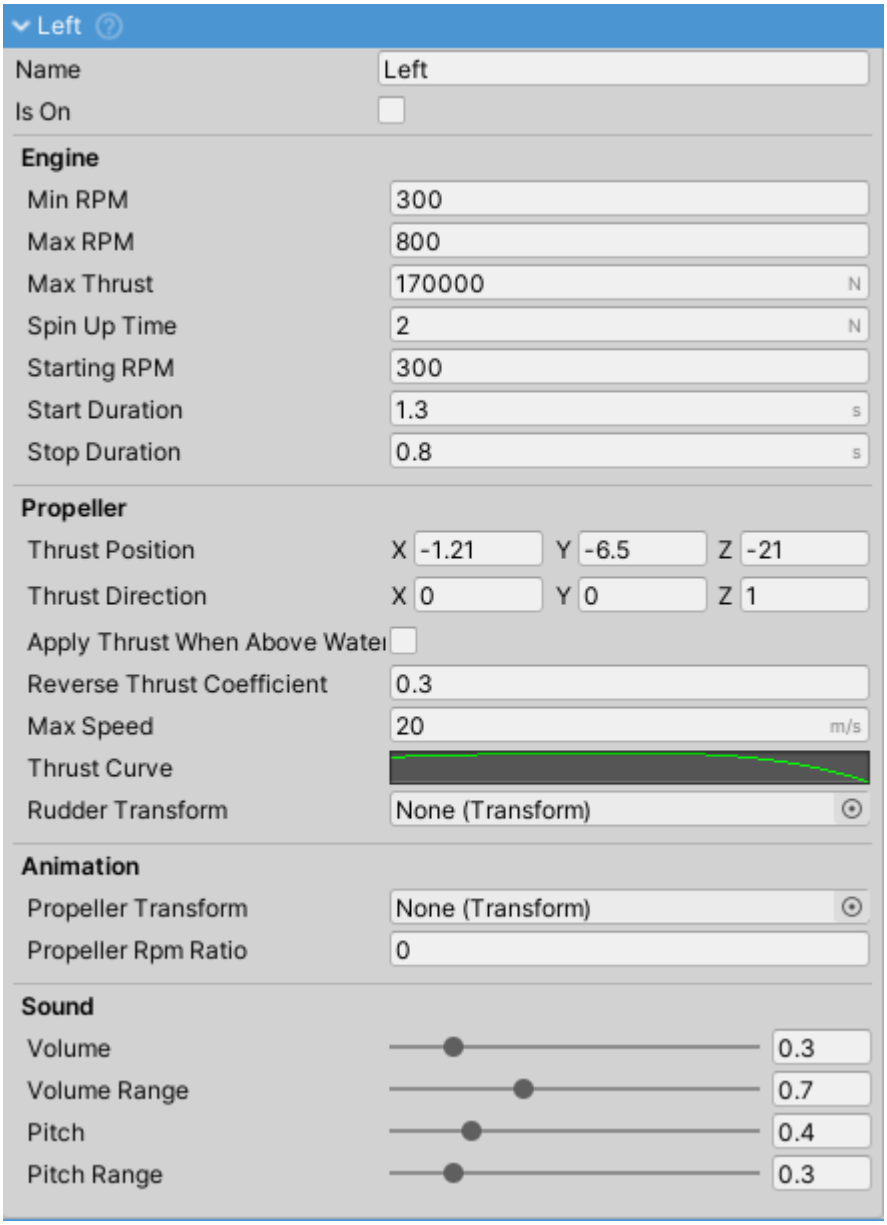
Fields

- `ballastChangeSpeed` - speed of change of the ballast mass, as a percentage of `maxBallastMass`. Controls how fast the submarine can take on or release ballast water.
- `maxBallastMass` - maximum ballast mass in kg that can be added to make the submarine sink. Higher values allow diving deeper and faster but require more time to surface.
- `keepHorizontal` - if enabled submarine will try to keep horizontal by shifting the center of mass.
- `keepHorizontalSensitivity` - sensitivity of calculation trying to keep the submarine horizontal. Higher number will mean faster reaction.
- `maxMassOffset` - maximum rigidbody center of mass offset that can be used to keep the submarine level.
- `DepthInput` - input property for depth control from -1 (surface) to 1 (dive). Positive values add ballast mass to sink, negative values reduce it to surface.

Setup

1. Add a `Submarine` component to the same GameObject that has `AdvancedShipController`.
2. Configure the ballast settings: set `maxBallastMass` to an appropriate value for your submarine and adjust `ballastChangeSpeed` to control how quickly the submarine changes depth.
3. Enable `keepHorizontal` if you want the submarine to automatically maintain a level orientation. Adjust `keepHorizontalSensitivity` and `maxMassOffset` to fine-tune the leveling behavior.
4. Ensure the vessel has a `VariableCenterOfMass` component. For more information, see NWH Common - Variable Center of Mass (https://nwhcoding.com/Common/manual/VariableCenterOfMass.html).
5. Set up depth input controls through your input system. The `DepthInput` property accepts values from -1 (surface) to 1 (dive).

Engine



An engine represents a single inboard or outboard engine. It handles power delivery, propeller rotation and engine sound. Each ship can have multiple engines.

> **Warning:** By default the thrust position of the engine is at [0,0,0]. Be sure to adjust this value to fit your ship.

Fields

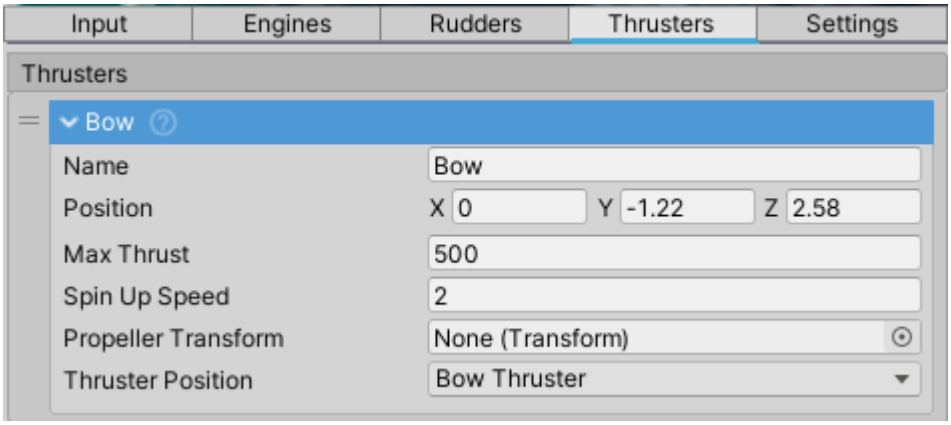
- `isOn` - is the engine currently on.
- `MinRPM`, `MaxRPM` - minimum and maximum RPM the engine can achieve. Stalling is not supported.
- `Max Thrust` - maximum thrust that the engine can generate at `MaxRPM`.
- `Spin Up Time` - time needed for the engine to reach `MaxRPM`.
- `Starting RPM` - RPM at which the engine will run while starting.
- `Start Duration`, `Stop Duration` - duration of engine starting and stopping. Influences for how long the start sound will be played.
- `Thrust Position` - position at which the thrust force will be applied. Local coordinates.
- `Thrust Direction` - direction in which the thrust will be applied. Local coordinates.
- `Apply Thrust When Above Water` - set to true if you want the propeller to work even when out of water. False by default.

- `Reverse Thrust Coefficient` - thrust coefficient when throttle is negative.
- `Max Speed` - ship speed in m/s at which propeller thrust drops to zero due to water flow matching propeller speed.
- `Thrust Curve` - thrust curve where the Y axis represents thrust percentage (of `Max Thrust`) and the X axis represents speed as a percentage (0-1) of `Max Speed`. This is essentially a speed/efficiency curve of a propeller.
- `Rudder Transform` - a transform representing the rudder. Can be used when the engine is outboard and the thrust should be applied in the direction of the rudder.
- `Propeller Transform` - a transform which will represent the propeller. Visual only, does not interact with water.
- `Propeller Rpm Ratio` - ratio between engine RPM and propeller RPM.

Setup

1. To make ship move an engine and propeller are needed. Ship Controller assumes that there is one propeller per engine. Add an engine under Engines tab and set the wanted values (hover over each value to see what it does). If thrust position is above the water thrust will not be applied (if `Apply Thrust When Above Water` is left unchecked).

Thruster

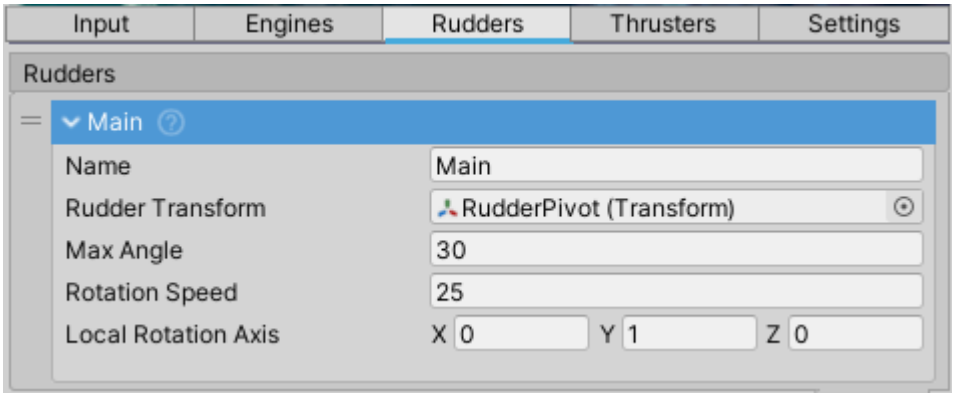


Thrusters can be used to move a ship without using the main engine(s). They can either apply thrust to the port or starboard side of the ship, typically used for maneuvering and station-keeping.

Fields

- `Position` - position at which the thrust will be applied.
- `Max Thrust` - maximum thrust in [N] which can be applied.
- `Spin Up Speed` - reaction time needed to reach `Max Thrust`
- `Thruster Position` - is it a bow or stern thruster. Determines input mapping.
- `Propeller Transform` - transform of the propeller that will be used as a visual representation of the thruster. Has to have Unity-correct rotation and pivot.
- `Propeller Rotation Direction` - rotation direction of `Propeller Transform`.
- `Propeller Rotation Speed` - rotation speed of the `Propeller Transform`.

Rudder



Rudder is used for steering the ship.

- Each rudder is a WaterObject and controls the ship through regular interaction with water.
- Invisible rudder can be used in case the visual one is too small. To achieve this parent a very thin Cube to the rudder, use WaterObjectWizard to set it up, resize it to the desirable scale and finally disable MeshFilter. This will result in a rudder that interacts with water but is not visible.

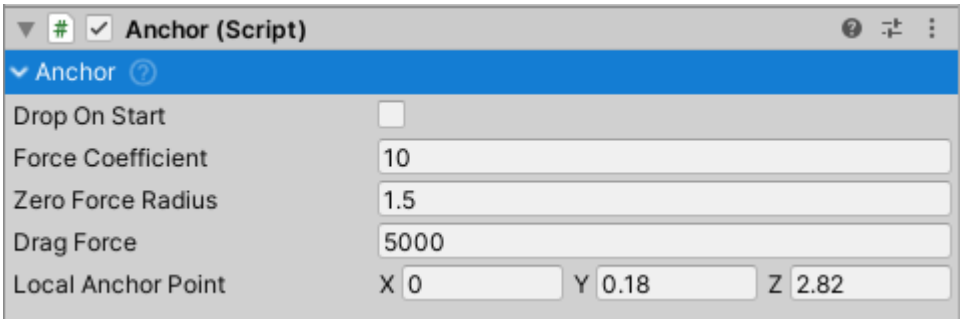
Fields

- `RudderTransform` - the transform that will be rotated.
- `MaxAngle` - maximum angle of rudder rotation to each side (e.g. +/- 30).
- `RotationSpeed` - rotation speed in deg/s for the rudder.
- `LocalRotationAxis` - the local axis around which the `RudderTransform` will be rotated.

Setup

1. Add a rudder to the ship if it does not already have one. This can be a primitive Cube scaled to a thin shape.
2. Assign the rudder transform (in this example the scaled cube) to the `Rudder Transform` field.
3. Add `WaterObject` component to the rudder so it too can interact with water.

Anchor



Anchor is a simple script that keeps a Rigidbody anchored to a point where the anchor was dropped through a physics joint. It approximates the behavior of a real anchor by keeping the object near the anchored position while allowing some movement, and can drag on the seafloor if forces exceed a threshold.

Fields

- `dropOnStart` - Should the anchor be dropped automatically at the start of the scene
- `localAnchorPoint` - Point in coordinates local to the object this script is attached to. Defines where on the ship the anchor chain is attached
- `zeroForceRadius` - Radius around the anchor in which the chain/rope is slack and no force will be applied. This allows small movements without resistance
- `forceCoefficient` - Coefficient by which the force will be multiplied when the object starts pulling on the anchor. Controls the strength of the anchoring force
- `dragForce` - Maximum force that can be applied to the anchor before it starts to drag along the seafloor

Public Properties and Methods

- `Drop()` - Drops the anchor. Sets the anchor position to the current anchor point position
- `Weigh()` - Weighs (retracts) the anchor, allowing the object to move freely
- `Dropped` - Property indicating whether the anchor has been dropped
- `IsDragging` - Property indicating whether the anchor is currently dragging on the seafloor (force exceeds dragForce)
- `AnchorPoint` - World space position of the anchor attachment point on the ship
- `AnchorPosition` - Current world position of the anchor itself

Input

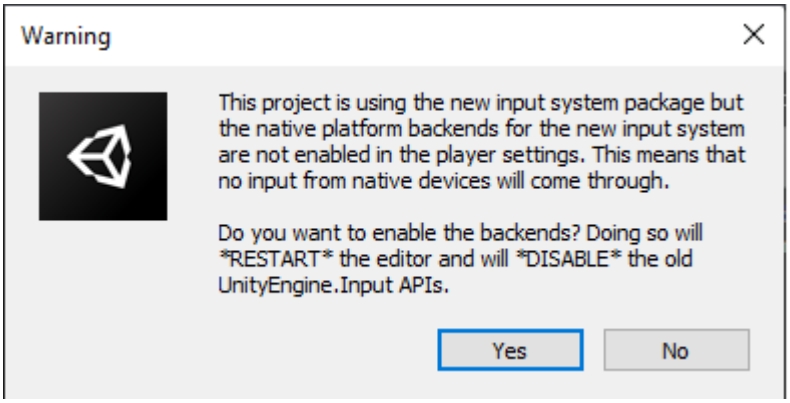
> **Note:** The Input System is part of NWH Common (https://nwhcoding.com/Common/). For detailed information about input providers, architecture, and creating custom providers, see NWH Common - Input System (https://nwhcoding.com/Common/manual/Input.html).

- Ships retrieve user input through `Input` class which retrieves input from active `InputProvider` and fills the `InputStates` struct with the retrieved data. * `InputProvider` s are split into *ShipInputProviders* and *SceneInputProviders*. *ShipInputProviders* relay ship input (throttle, brakes, etc.) while *SceneInputProviders* take care of scene input (ship changing, camera changing, camera movement and the rest of the inputs not directly related to ship. One of each needs to be present (e.g. `InputSystemShipInputProvider` and `InputSystemSceneInputProvider`).
- Multiple different `InputProvider` s can be present in the scene (v1.0 or newer required). E.g. InputSystemProviders and MobileInputProviders can be used in the same scene. The resulting input will be a sum of inputs from all `InputProvider` s in case of numeric inputs and logical OR operation of all inputs in case of boolean inputs.
- Input is stored inside `InputStates` object and can be copied over from one ship to another.
- To manually set the `InputStates` make sure `Auto Settable` is set to false.

All input providers inherit from either `ShipInputProviderBase` or `SceneInputProviderBase` , but differ in their implementation.

Input System Warning

When importing the asset for the first time this message will pop up:



Select `Yes` to enable the new Input System. The demo scenes rely on InputSystem.
If a message *This Unity Package has Package Manager dependencies.* appears, click `Install/Upgrade` .

Available Bindings

Ship Input Provider Bindings

> **Warning:** Out of the box gamepad bindings are only available for InputSystem.

Name	Type	Keyboard Defaults	Gamepad Defaults	Description
Steering	axis [-1,1]	A/D	Left Stick - Left/Right	Steering/rudder. Positive right.
Throttle	axis [-1,1]	S/W	Left/Right Trigger	Throttle. Positive forward.
Throttle2	axis [-1,1]			Throttle 2.
Throttle3	axis [-1,1]			Throttle 3.
Throttle4	axis [-1,1]			Throttle 4.
BowThruster	axis [-1,1]	Q/E	Left Stick - Left/Right	Bow thruster input.
SternThruster	axis [-1,1]	Z/C		Stern thruster input.
SubmarineDepth	axis [-1,1]	K/I		
EngineStartStop	Button	E		
Anchor	Button	T		

Scene Input Provider Bindings

Name	Type	Keyboard Defaults	Gamepad Defaults	Description
ChangeCamera	button	C	Start	Changes camera.
CameraRotation	2D axis	Mouse Delta	Right Stick	Controls camera rotation.
CameraPanning	2D axis	Mouse Delta	Right Stick	Controls camera panning.
CameraRotationModifier	button	Mouse - LMB	Right Stick Press	Enables camera rotation.
CameraPanningModifier	button	Mouse - RMB	Left Stick Press	Enables camera panning.
CameraZoom	axis	Mouse - Scroll	D-Pad Up/Down	Camera zoom in/out.
ChangeVehicle	button	V	Select	Change vehicle or enter/exit vehicle.
FPSMovement	2D axis	WASD	Left Stick	Demo FPS controller movement.
ToggleGUI	button	Tab		Toggles demo scene GUI.

Input Manager (old/classic)

- Type of `InputProvider` for handling user input on desktop devices through keyboard and mouse or gamepad.
- Uses classic/old Unity Input Manager. It is recommended to use the Unity's new Input System instead for new projects.

> **Note:** InputSystem package is required. The old/classic Unity `InputManager` is no longer supported.

Installation

When first importing Dynamic Water Physics 2 the project will be missing required bindings. There are two ways to add those:

1. Manually adding each entry to the *Project Settings* => *Input* following the input bindings table.
2. Copying the contents of *InputBindings.txt* and appending them to the contents of the `[UnityProjectPath]/ProjectSettings/InputManager.asset` file. To do so:
    - Close Unity.
    - Open *InputManager.asset* in Notepad/Notepad++/Visual Studio or any other text editor of your choice.
    - Copy the contents of the provided InputBindings.txt file (*Scripts* => *ShipController* => *Input* => *InputProviders* => *InputManagerProvider* => *InputBindings.txt*) and paste them at the end of the *InputManager.asset*. Make sure there are no empty lines between the existing content and the pasted content. Also make sure that all the indents are correct (Unity will detect end of file if indent is off). Save the file.
    - Open Unity. Check *Project Settings* => *Input*. The input bindings for Dynamic Water Physics will appear towards the bottom of the list.

Scene Setup

To set up InputManager-based input in the scene add the following components to the scene:

1. 'InputManagerShipInputProvider'
2. 'InputManagerSceneInputProvider'

Any ship that is present in the scene will now receive input from these providers.

Input System (new)

- InputSystem v1.0 or higher is required. This is available in Unity 2019.3 or newer.

> **Warning:** When using DS4Windows, InputSystem will detect button presses twice.

Installation

- Install 'Input System' package through Window => Package Manager
- Under Edit => Project Settings => Player => Other Settings, set `Active Input Handling` to `Input System Package (New)` or `Both` - the latter in case your project still uses `UnityEngine.Input` somewhere.

Scene Setup

- Add `InputSystemShipInputProvider` and `InputSystemSceneInputProvider` to any object in your scene.
- Default bindings can be modified by double clicking on `.inputactions` files. `Save Asset` must be clicked for the changes to take effect.

Mobile Input Provider

- Add `MobileShipInputProvider` and `MobileSceneInputProvider` to the scene.
- Create a few UI => Button objects inside your canvas. Make sure that they are clickable.
- Remove the `UnityEngine.UI.Button` component and replace it with `MobileInputButton`. `MobileInputButton` inherits from `UnityEngine.UI.Button` and adds `hasBeenClicked` and `isPressed` fields which are required for `Mobile Input Provider`
- Drag the buttons to the corresponding fields in the `MobileShipInputProvider` and `MobileSceneInputProvider` inspectors. Empty fields will be ignored.

Scripting

Retrieving Input

Since v1.0 multiple `InputProvider` s can be present in the scene, meaning that their input has to be combined to get the final input result. To get the combined input use:

```
float throttle = InputProvider.CombinedInput(i => i.Throttle());
bool engineStartStop = InputProvider.CombinedInput(i => i.EngineStartStop());
```

Or to get the input from individual `InputProvider` s (say to find out if a button was pressed on a keyboard):

```
float throttle = InputProvider.Instances[0].Throttle;
```

When using input generated by code (i.e. AI) it is usually handy to have access to a single axis throttle/brake. This can be done like so:

```
shipController.input.Throttle = 0.5f;
shipController.input.Throttle = -0.5f;
```

> **Warning:** `shipController.input.states.throttle` is equal to `shipController.input.Throttle` . The latter is just a getter/setter for convenience.

Manually Setting Input

Input in each ship is stored in `InputStates` struct:

```
myShipController.input.states
```

In case input should not be retrieved from user but from another script - as is the case when AI is used - `AutoSettable` should be set to `false` . This will disable automatic input fetching from the active `InputProvider` .

Input now can be set from any script:

```
myShipController.input.Horizontal = myFloatValue; // Using getter/setter.
myShipController.input.states.horizontal = myFloatValue; // Directly accessing states.
```

Custom InputProvider

If a custom `InputProvider` is needed it can easily be written. Custom `InputProviders` allow for new input methods or for modifying the existing ones. E.g. if the `MobileInputProvider` does not fit the needs of the project a copy of it can be made and modifications done on that copy. That way it will not get overwritten when the asset is updated.

Steps to create a new `InputProvider` :

- Create a new class, e.g. `ExampleInputProvider` and make it inherit from `InputProvider` class:

```
public class ExampleInputProvider : InputProvider {}
```

- Implement missing methods. Most IDEs can do this automatically.
- The required methods are *abstract* and will need to be implemented. There are also *virtual* methods such as `ToggleGUI()` which are optional and will be ignored if not implemented.
- Methods that are not used should return `false`, `0` or `-999` in case of `ShiftInto()` method.

Input System Provider

- Ships retrieve user input through `Input` class which retrieves input from active `InputProvider` and fills the `InputStates` struct with the retrieved data.
    - Multiple different `InputProvider` s can be present in the scene (v1.0 or newer required). E.g. InputSystemProvider and MobileInputProvider can be used in the same scene. The resulting input will be a sum of inputs from all `InputProvider` s in case of numeric inputs and logical OR operation of all inputs in case of boolean inputs.
- Input is stored inside `InputStates` object and can be copied over from one ship to another.
- To manually set the `InputStates` make sure `Auto Settable` is set to false.

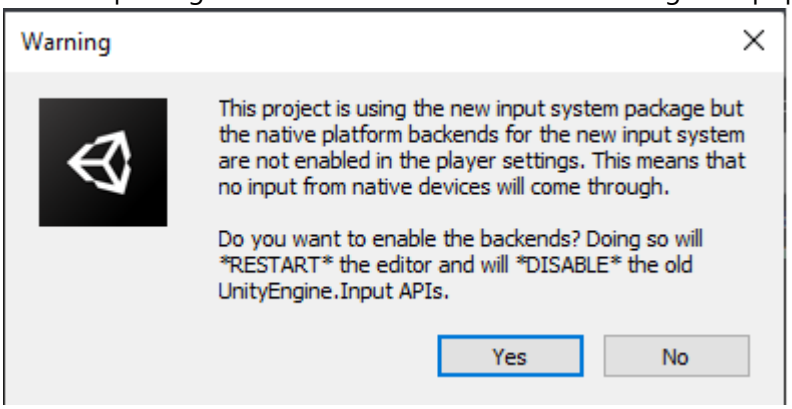
All input providers inherit from abstract class `InputProvider` but have different implementations:

- InputManagerProvider (Input.html#input-manager-oldclassic) - for handling keyboard and mouse or gamepad input through InputManager.
- InputSystemProvider (Input.html#input-system-new) - same as InputManagerProvider but for new Unity Input System (https://docs.unity3d.com/Manual/com.unity.inputsystem.html).
- MobileInputProvider (Input.html#mobile-input-provider) - GUI based input.

Check the `Scripting` section below on how to make a custom InputProvider or set up AI.

Input System Warning

When importing the asset for the first time this message will pop up:



Select *Yes* to enable the new InputSystem. The demo scenes rely on InputSystem.

Available Bindings

> **Warning:** Out of the box gamepad bindings are only available for InputSystem.

Name	Type	Keyboard Defaults	Gamepad Defaults	Description
Ship Bindings				
`Steering`	axis [-1,1]	A/D	Left Stick - Left/Right	Steering.
`Throttle`	axis [-1,1]	W/S	Left Stick - Up/Down, Left/Right Trigger	Throttle.
`BowThruster`	axis [-1,1]	Q/E		Bow thruster.
`SternThruster`	axis [-1,1]	Z/C		Stern thruster.
`SubmarineDepth`	axis [-1,1]	K/I		Submarine depth.
`EngineStartStop`	button	E	Square (PS) / X (Xbox)	Start or stop the engine.
`Anchor`	button	T	Triangle (PS) / Y (Xbox)	Drop or weight the anchor.
Camera Bindings				
`ChangeCamera`	button	C	Start	Changes camera.
`CameraRotation`	2D axis	Mouse Delta	Right Stick	Controls camera rotation.
`CameraRotationModifier`	button	Mouse - LMB	Right Stick Press	Enables camera rotation.
`CameraZoom`	axis	Mouse - Scroll	D-Pad Up/Down	Camera zoom in/out.
Scene Bindings				
`ChangeShip`	button	V	Select	Change ship.
`ToggleGUI`	button	Tab		Toggles demo scene GUI.

Input Manager (old/classic)

Input System (new)

Mobile Input Provider

- Add `MobileInputProvider` to the scene.
- Create a few UI => Button objects inside your canvas. Make sure that they are clickable.
- Remove the `UnityEngine.UI.Button` component and replace it with `MobileInputButton`. `MobileInputButton` inherits from `UnityEngine.UI.Button` and adds `hasBeenClicked` and `isPressed` fields which are required for `Mobile Input Provider`
- Drag the buttons to the corresponding fields in the `MobileInputProvider`. Empty fields will be ignored.

Scripting

Retrieving Input

Multiple `InputProvider`s can be present in the scene, meaning that their input has to be combined to get the final input result. To get the combined input use:

```
float throttle = InputProvider.CombinedInput(i => i.Throttle());
bool engineStartStop = InputProvider.CombinedInput(i => i.EngineStartStop());
```

Or to get the input from individual `InputProvider`s (say to find out if a button was pressed on a keyboard):

```
float throttle = InputProvider.Instances[0].Throttle;
```

When using input generated by code (i.e. AI) it is usually handy to have access to a single axis throttle/brake. This can be done like so:

```
shipController.input.Throttle = 0.5f;
```

> **Warning:** `shipController.input.states.throttle` is equal to `shipController.input.Throttle`. The latter is just a getter/setter for convenience.

Manually Setting Input

Input in each ship is stored in `InputStates` struct:

```
myShipController.input.states
```

In case input should not be retrieved from user but from another script - as is the case when AI is used - `AutoSettable` should be set to `false`. This will disable automatic input fetching from the active `InputProvider`.

Input now can be set from any script:

```
myShipController.input.Horizontal = myFloatValue; // Using getter/setter.

myShipController.input.states.horizontal = myFloatValue; // Directly accessing states.
```

Custom InputProvider

If a custom `InputProvider` is needed it can easily be written. Custom `InputProviders` allow for new input methods or for modifying the existing ones. E.g. if the `MobileInputProvider` does not fit the needs of the project a copy of it can be made and modifications done on that copy. That way it will not get overwritten when the asset is updated.

Steps to create a new `InputProvider`:

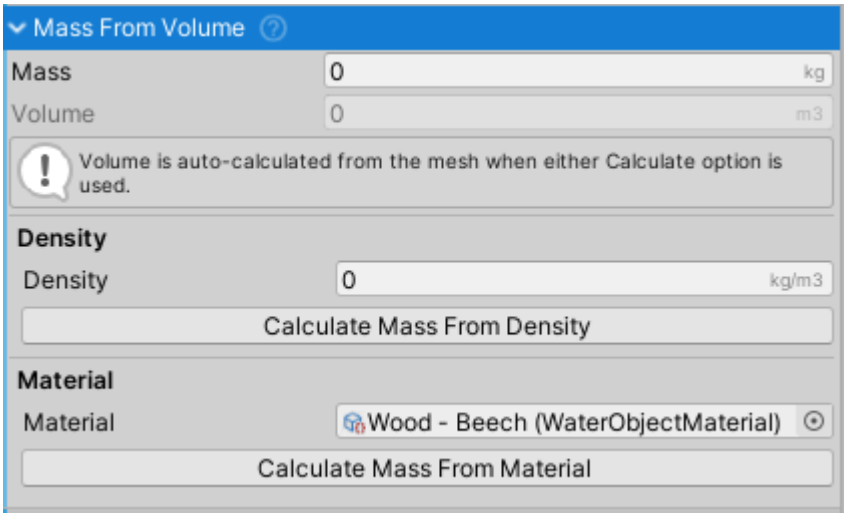
- Create a new class, e.g. `ExampleInputProvider` and make it inherit from `InputProvider` class:

```
public class ExampleInputProvider : InputProvider {}
```

- Overwrite required methods.

[!redirect "Input.md"]

Mass From Volume



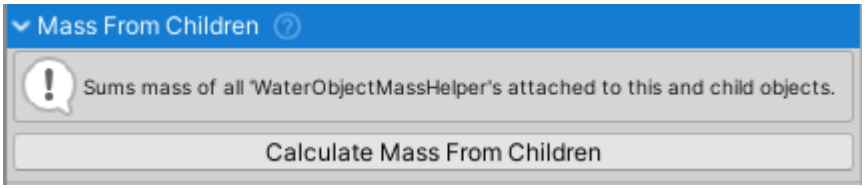
`MassFromVolume` is a helper script that calculates object's mass from volume of the mesh and the density.

- Volume of the mesh is calculated automatically.
- Can be used together with MassFromChildren (MassFromChildren.html) to calculate mass of complex objects (objects having more than one child WaterObject (WaterObject.html)).

Usage

- Attach the script to the object that has the mesh.
- The `Rigidbody` on the same object will have its mass set automatically.
- Adjust the `Density` field to control the mass calculation (mass = volume × density).

Mass From Children



A helper script for determining mass of a Rigidbody from the children. It sums the masses of all the children that have MassFromVolume (MassFromVolume.html) scripts attached. This eliminates the need for guessing the mass of the object.

Usage

- Attach the script to the parent object. That object has to contain the `Rigidbody` component.
- Make sure that at least one child has MassFromVolume (MassFromVolume.html) attached or the result will be 0 and ignored.
- Press *Calculate Mass From Children*. This will calculate and set the `Rigidbody` 's mass.

Multiplayer

Dynamic Water Physics 2 officially supports the following networking solutions:

- Photon Unity Networking (PUN) 2 Setup Guide (PhotonUnityNetworking2.html)
- Mirror Setup Guide (Mirror.html)

Mirror

Setup guide for Dynamic Water Physics 2 multiplayer using Mirror (https://assetstore.unity.com/packages/tools/network/mirror-129903) framework.

The guide shows how to set up `AdvancedShipController` for multiplayer. If setting up just a simple `WaterObject` use the `NetworkRigidbody` as the `WaterObjects` are just simple Rigidbodies and do not need any advanced synchronization.

Mirror demo scene is included with the asset.

> **Warning:** Make sure that the scene contains required InputProviders for the input system that is being used. More about Input here (Input.html).

Setting up the scene

- Create a new *GameObject* called 'NetworkManager'.
- Add the following components to the created object:
  - *NetworkManager*
  - *NetworkManagerHUD*
  - *TelepathyTransport* (or other transport if you are not using Telepathy). In newer versions of Mirror *KcpTransport* is the default.

> **Warning:** The scene should contain required InputProviders for the input system that is being used. More about Input here (Input.html).

Setting up the vehicle

- Double click on Mirror.unitypackage (*NWH\Dynamic Water Physics 2\OptionalPackages\Multiplayer*) to import the required assets.
- Add *MirrorMultiplayerShip* component to the ship. This will also add necessary Mirror components automatically.
- Make sure that *NetworkTransform*'s and *NetworkRigidbody*'s *Client Authority* fields are ticked.
  - Set *Network Sync Interval* to 0.05 (20Hz). Default value is 0.1 (10Hz) and is too low for physics objects and will result in stuttering.
  - Set *Compress Rotation* to *None*.
  - Save the vehicle as a prefab and assign it to `NetworkManager` 's *Player Prefab* field.

Photon Unity Networking2

Setup guide for Dynamic Water Physics 2 multiplayer using Photon Unity Networking 2 (https://www.photonengine.com/) framework.

PUN2 demo scene is included with the asset.

> **Warning:** Check the Photon Unity Networking 2 setup with provided Photon demo scenes (e.g. slot cars) to verify that the Photon setup has been done correctly.

Adding PUN2 to the scene

- Create a new GameObject and name it *PhotonManager*.
- Add *OnJoinedInstantiate* and *ConnectAndJoinRandom* to the newly created object.
- Set position offset to some higher value (e.g. >10).
- Set *Auto Connect* to *true*.

> **Warning:** The scene should contain required InputProviders for the input system that is being used. More about Input here (Input.html).

Adding PUN2 to the ship

- Add *PhotonMultiplayerShip* component to your ship. This will automatically add other necessary scripts.
- Make sure that under *Photon View* component *Observe option* is not set to ***Off***. Any other option can be used.
- Observed Components should look as in the following image. If any component is missing from the list add it.

Instantiating

- Objects that will be instantiated by *OnJoinedInstantiate* need to be placed under *Resources* folder. If there is no such folder, create one; its location does not matter. Drag the ship prefab to that folder.
- Assign the vehicle prefab under *Prefabs To Instantiate* of *OnJoinedInstantiate* script.
- Press Play. Vehicle will instantiate after a few seconds.
- Build the project and open two windows side by side to test it out.

Upgrade Notes

2.x to 3.x

- Clean import through Package Manager is required.
- Depends on NWH Shared / Common Library asset.
- Required use of URP or HDRP for sample assets, core asset does not depend on specific render pipeline.
- 3rd part water integration is now done through the Package Manager Samples tap for the Dynamic Water Physics 2. Adjusting asmdefs and adding scripting define symbols is no longer needed.

2.6 to 2.7

- Clean import required.
- Required adjustment of CenterOfMass on VariableCenterOfMass script since the CoM is now set as an absolute value instead of an offset.

2.5.5 to 2.5.6

- Clean import required.
- WaterDataProviders moved from renaming the files to remove the .txt extension to using Scripting Define Symbols. Check the WaterDataProvider documentation for the new instructions.

2.5.3 to 2.5.4

- Clean import required.

2.5.2 to 2.5.3

- VariableCenterOfMass `Center Of Mass Offset` and `Inertia Scale` might need re-adjusting due to changes in the VariableCenterOfMass script. Effective inertia in the new version will be ~3 times lower than the calculated value in the previous version. Due to the bug fix that affected center of mass calculation, the center of mass with the new version will be different if any `IMassAffector` s have been attached.

2.4.x to 2.5

- `WaterObjects` will need to have their simulation meshes re-generated by pressing *Water Object => Update Simulation Mesh*.
- `WaterObjectManager` has been removed and the settings have moved to individual `WaterObject` s.
- `WaterDataProvider` now requires a `Collider` attached to the same `GameObject` with `Is Trigger` enabled. Objects that are inside this trigger volume will use the data from that `WaterDataProvider` .
- `WaterObject` can now work without any `WaterDataProvider` s present. `defaultWaterHeight` , `defaultWaterNormal` and `defaultWaterFlow` will be used in this case.

2.4.1 to 2.4.2

- Asset now uses single assembly definition placed in the root directory of the asset, named NWH.DWP2. Any references that need to be added to external water assets now have to be added here instead of the NWH.DWP2.WaterData assembly definition.

2.3 to 2.4

- Clean import is required.
- `Project Settings > Player > API Compatibility Level` needs to be set to `.NET 4.x`.
- `CenterOfMass` script was replaced with `VariableCenterOfMass`.
- Since 2.4 the asset uses assembly definition files. This might cause issues with projects that do not use .asmdef files and errors along the lines of "The type or namespace 'x' could not be found...". The solution is to either make the project use .asmdef files or to delete all the .asmdef files inside the this asset's directory. This is because the whole project needs to use assembly definitions or not use them at all. More about this here: Assembly Definitions (QuickStart.html)
- Any 3rd party water asset will need to have .asmdef added (if not already present such as with Crest) and referenced inside the NWH.DWP2.WaterData assembly definition (NWH > Dynamic Water Physics 2 Scripts > WaterDataProvider). Alternative is to remove all assembly definition (.asmdef) files from the asset as mentioned in the step above.

2.2 to 2.3

Due to asset restructure it is recommended to do a clean import of DWP2 when upgrading from 2.2 to 2.3.

2.1 to 2.2

Due to the number of changes, changed script names and locations it is recommended to do a clean import of DWP2 when upgrading from 2.1 to 2.2.

The update from 2.1 to 2.2 is a large one. Things to note:

- `WaterObject` material settings have been moved to a separate script: `MassFromVolume` and now requires button press (or a function call) to apply the mass.
- `WaterObject` materials are now `ScriptableObject`s instead of a static list.
- Cameras, `VehicleChanger` and other scene scripts are now shared between NWH Vehicle Physics 2 and Dynamic Water Physics 2. This improved the quality of DWP2 scripts but unfortunately it means that these scripts will need to be re-added and reconfigured.
- Input System needs to be installed and *Project Settings* > *Input Handling* needs to be set to both for demo scenes to work properly.
- Version 2.2 has a different approach to support for water systems. This has been greatly simplified but a re-setup will be needed. Check Water Asset Setup Guide (WaterAssets.html) for more info.
- Input in 2.2 works differently. Check Input (Input.html) page for more info.
- If using InputManager (old/classic Unity input) make sure to add `Throttle` and `Steering` axes to *Project Settings* => *Input Manager* as `Horizontal` and `Vertical` are no longer used.

1.x to 2.x

Due to a large number of changes between DWP1 and DWP2 most, if not all, objects will need to be reconfigured. DWP1 and DWP2 can co-exist in the same project due to different namespaces. Some scripts might have the same name in the AddComponent dialog if both versions are present.

Warning: Due to Unity Asset Store importer not recognizing different namespace scripts as different files you can not directly import one version from the Store while the other is present. Download and import the new version into an empty project and then copy it over manually to be able to use both assets side by side.

This is happening because some scripts retained their names in the new version and despite having completely different namespace Unity will still think they are the same script (only the filename is checked) and overwrite the one already inside the project. This means that if DWP2 is imported after DWP1, some of the scripts from DWP1 will be overwritten by those in DWP2.

Import Steps

The steps to import DWP2 with DWP1 present in the project:

1. Make sure that your project is using Unity 2018.4 or newer.
2. Create a new project using Unity 2018.4 or newer.
3. Import DWP2 from Asset Store into the new project.
4. Close the new project - ignore any errors if present (check README later for those).
5. Open file explorer and copy the DWP2 folder from the new project into the project you are upgrading.

You could of course just delete DWP1 and import DWP2 but that will cause a lot of missing script errors on the existing objects, and it would make the upgrade method described below impossible.

Water Object Manager

DWP2 requires exactly one WaterObjectManager to be present in the scene. It does not matter to which object you attach it - usually it would be something like 'SceneManager' - it just has to be present as it does all the physics calculations.

WaterObject

WaterObject (previously known as FloatingObject) editor has been rewritten and therefore the old one can not be used with the new system. Fastest way to upgrade would be removing the old Floating Object script, together with the old water effects system, and adding WaterObjectWizard component which will automatically configure the object. It is possible to run WaterObjectWizard on multiple objects at the same time.

Before upgrading it is always a good idea to make a backup.

Step-by-step:

- Filter out WaterFX objects and remove WaterFX components
- Do the same for FloatingObject. Do not deselect the objects afterwards.
- Add to the same objects that you have filtered WaterObjectWizard component
- Tick the option for WaterParticleSystem (replaces WaterFX) if needed and run the wizard
- Add WaterObjectManager to any object in the scene (usually SceneManager or similar)

The scene is now upgraded. The settings will not carry over from WaterFX and FloatingObject to the new scripts so some manual tweaking is needed.

Changelog

v3.0

04/11/2025

- Upgraded to Unity 6000
- Migrated to URP as default render pipeline
- Moved to Package Manager format with Runtime and Samples~ folders
- Water integration now done through samples, asmdefs and scripting define symbols no longer need to be set up
- New DocFX-based documentation with auto-generated API Reference
- Improved code documentation and comments throughout

v2.22

09/09/2024

- Updated to Unity 2022.
- Updated the RaycastWaterDataProvider to avoid obsolete warnings.
- Updated the Vehicle class with input deadzone and other commonly used constants.
- Updates to CrestWaterDataProvider for Crest 5.

v2.21f2

22/01/2024

- Renamed floating origin to shifting origin because of copyright claims.

v2.21f1

22/01/2024

- Documentation update.

v2.21

25/11/2023

- CrestWaterDataProvider cleanup and fixed flow issue with Crest (@Kyleli).
- Fixed OnExitWaterDataProvider causing index out of range issue.

v2.20

26/06/2023

- Fixed variable CoM combined mass affector is never applied.
- Fixed first mass affector is skipped.

v2.19

09/06/2023

- Editor file restructuring to allow the asset to work without assembly definitions. Clean import required!

v2.18

29/05/2023

- `VariableCenterOfMass` Fixed null issue when the GameObject is disabled.
- `Vehicle` onActiveVehicleChanged now passes previous and current vehicles as params.
- `WaterObject` Fixed the issue where when there are no WaterDataProviders active the WaterObject would remember the data from the last WaterDataProvider, instead of falling back to default.

v2.17

22/05/2023

- `Vehicle` Added Vehicle.ActiveVehicle which replaces VehicleChanger.ActiveVehicle. This removes the dependency of multiple scripts on VehicleChanger. Also makes the demo scripts work without it.
- `Vehicle` Added 'isPlayerControllable' option. If this is set to false the vehicle will never be set as Vehicle.ActiveVehicle. Useful for trailers and other passive vehicles as the ActiveVehicle is always the last enabled vehicle.

v2.16

19/05/2023

- Fixed intermittent NaN error caused by WaterObject.
- Fixed SailController lift force direction, introduced in 2.15.

v2.15

16/05/2023

- Updated KWSWaterDataProvider to work with the latest KWS version.
- Fixed SailPreset build bug.
- Fix SailController lift force direction.
- Added SailController separate drag and lift force direction gizmos.

v2.14

11/05/2023

- Added SailPresets to SailController for better adjustability. These replace the previous lift and drag coefficients and the fixed force calculation.
- Tab now toggles the demo UI.

v2.13

03/05/2023

- Cleaned up and commented out the sailing code. Added custom editors for all the sailing scripts.
- Rework of the Submarine and Sink components. Both are now just VariableCenterOfMass MassAffectors, meaning they act as variable weight/ballast attached to the ship.
- Transform position of the Sink component itself now determines the center of mass offset point / point of water ingress.
- Submarine depth input now changes the relative depth instead of setting the absolute depth value. This means that negative input means deeper, positive input means shallower.
- VariableCenterOfMass changes, including the removal of the unused centerOfMassOffset and bug fixes.

v2.12

02/05/2023

- Added sail controller early preview.
- Added Sailboat to the demo.
- Fixed WaterObjectWizard null bug.

v2.11

25/04/2023

- Wake/Sleep has been removed and replaced with the standard Behaviour enable/disable.
- Common code update.

v2.10

15/04/2023

- “VehicleReflectionProbe” and “FollowVehicleState” are now in Common code and available in DWP2.
- Fixed GC issue with Mathf.Min.
- Exposed “Awake On Start” option in Settings.
- Added underwater fog to the demo.
- Fixed WaterObject spamming prefab Overrides.

v2.9

12/04/2023

- Common code update.
- ‘registerWithVehicleChanger’ and ‘autoFindVehicles’ have been removed as they add unneeded complexity. Use VehicleChanger.Instance.RegisterVehicle() and DeregisterVehicle() after spawning a vehicle instead.
- Vehicles no longer get woken up automatically if VehicleChanger is present and active.

v2.8

10/04/2023

- Common code update.
- Updated multiplayer code.

v2.7

10/04/2023

- Common code update.

v2.6.2

28/05/2022

- Common code update.

v2.6.1

09/05/2022

- Added KWS support.
- Fixed very small triangles getting ignored by simulation.

v2.6.0

06/04/2022

- Added Mirror and PUN2 multiplayer support for AdvancedShipController.
- Added StartOnThrottle option to Engine.

v2.5.6

05/04/2022

- Changed WaterDataProviders from file rename approach to Scripting Define Symbols approach.
- Missing or incorrectly configured trigger collider on the WaterDataProvider will now get automatically added if missing, throwing just a warning instead of error.
- Demo UI changes.
- Improved the documentation regarding WaterDataProviders.

v2.5.5

24/03/2022

- Fixed ReorderableList not working properly in the latest versions of Unity 2021 and 2022.
- Fixed stabilize roll and stabilize pitch not working for negative angles.

v2.5.4

22/01/2022

- Added PostProcessing to the demo scene.
- WelcomeMessage fix.

v2.5.3

05/11/2021

- VariableCenterOfMass improvements and bug fixes.

v2.5.2

04/11/2021

- Moved to Unity 2020 LTS.
- Fixed VariableCenterOfMass calculating center of mass incorrectly.
- Fixed issues with assembly definitions.

v2.5.1

01/11/2021

- Fixed issue with simulation mesh not serializing properly.
- Fixed issues with Crest and Lux WaterDataProviders.
- Returned the default Skin Drag Coefficient value to 0.01 (same as pre 2.5).
- Removed unused 'Water Density' field from WaterDataProvider.

v2.5.0

30/10/2021

- Rewritten the WaterObject to remove the need for WaterObjectManager and Synchronize().
- Removed WaterObjectManager.

- Multiple WaterDataProviders can now be present in the scene. Triggers are used to detect the currently active provider.
- Other bug fixes and improvements.

v2.4.3

08/06/2021

- Moved WaterObject gizmos to WaterObject from WaterObjectDebug. Changes to make Gizmos more similar to the ones on the initial release.
- Fixed center of mass creep while the Rigidbody is marked as kinematic.
- Fixed WaterObjectWizard using legacy CenterOfMass script.
- Fixed WaterObject trying to access shared data before being initialized resulting in an out of bounds exception.
- Fixed DragObject NullReferenceException issue.
- Fixed disabled triangle state getting overwritten resulting in disabled objects still being simulated.
- Fixed multiple issues with WaterObjectManager.Synchronize().
- Optimized WaterObjectManager.Schedule().

v2.4.2

04/05/2021

- Removed unused Package Manager dependencies.
- Moved WaterDataProvider to WaterObject assembly. This was required due to the circular references between RAMWaterDataProvider and WaterObject. This will require moving the assembly references for Crest, Lux or other 3rd party water assets to NWH.DWP2 assembly definition.
- WaterObjectSetupWizard now adds VariableCenterOfMass instead of the obsolete CenterOfMass script.
- Fixed MassFromChildren not setting VariableCenterOfMass mass.
- Fixed ReorderableList crashing Editor in Unity 2021.

v2.4.1

14/03/2021

- Removed requirement for .NET 4.x.
- Fixed the import issues introduced with the previous version.

v2.4

09/03/2021

- Added .asmdefs (clean import required).
- (Re)Introduced “Finish Jobs In One Frame” option to the WaterObjectManager.
- Made InputActions path relative.
- Fixed WaterParticleSystem water line using inactive/incorrect triangle edges.
- Fixed “Weld Colocated Vertices” being ignored when “Simplify Mesh” option is disabled.
- Fixed WaterObject gizmos not always being drawn properly.
- Fixed deleting WaterObject causing errors.
- Fixed depth at the center of a split triangle sometimes registers as too high.

v2.3.3

05/02/2021

- CameraMouseDrag can now be used for both 1st and 3rd person view (including rotation, panning and camera shake) and will replace other cameras in the future versions as an universal camera.
- Reworked Submarine script and added a PID controller for better depth control.

- Added option to synchronize WaterObjects in current scene or all loaded scenes.
- Fixed editor division by zero when no WaterObjects present.
- Fixed missing 'using Unity.Burst;' inside WaterTriangleJob.
- Fixed Burst compatibility on newer versions of Unity.

v2.3.2

17/12/2020

- Added per-object simulation settings.
- Added multiple throttle bindings.
- Fixed inspector entering infinite loop on Unity 2020.2 when ReorderableList is drawn.

v2.3.1

23/11/2020

- Fixed import bug introduced in v2.3.

v2.3

16/11/2020

- Asset restructure to allow for seamless import of multiple NWH assets into the same project. Clean import needed.

v2.2.4

28/10/2020

- Fixed 'Simulate Water Normals' option causing issues with RigidbodyWaterDataProvider when used with a plane.
- Fixed two audio listeners being active at the same time when changing a vehicle.
- Updated NWH/Common directory to match NVP2.
- Fixed steering value range in editor.
- Fixed sound sources not being visible in the engine inspector.

v2.2.3

23/08/2020

- Fixed foam particles not being generated by WaterParticleSystem.

v2.2.2

14/08/2020

- Fixed missing path error related to SceneInputActions file.

v2.2.1

01/08/2020

- Fixed full stack debugging being forced on by WaterObjectManager decreasing editor performance.
- Fixed issue with CrestWaterData provider when DWP_Crest was defined as a leftover from v2.1 and older.

v2.2

17/07/2020

- Moved editor drawing to NWH.NUI for all scripts.

- Added support for R.A.M.
- Added option to weld simulation mesh vertices.
- WaterDataProviders are now MonoBehaviors and have been simplified for easier implementation of custom providers.
- Cameras, VehicleChanger and other scene scripts are now the same between NWH Vehicle Physics 2 and Dynamic Water Physics 2. This enables use of both in the same scene.
- Reworked input to use InputProvider system from NWH Vehicle Physics 2; both InputManager and InputSystem are now supported. Customizing input is now much easier.
- Added support for water normals for water systems that can provide this info.
- WaterObject materials are now ScriptableObjects instead of a static list.
- Add roll stabilization.
- Improved code related to water object simulation for better performance.
- Added support for Collider based water.
- Add “IsTouchingWater” to WaterObject.
- Fixed water flow calculation.
- Fixed WaterObjects not being detected in additively loaded scenes.
- Other minor bug fixes and changes.

Support

For any questions, bug reports, issues or general inquiries, email us (mailto:nwhcoding@gmail.com).