



**Crest API**

*Release 5.7.0*

**Wave Harmonic**

**Jan 23, 2026**

# TABLE OF CONTENTS

<b>1 Crest</b>	<b>1</b>
1.1 AbsorptionLod . . . . .	1
1.2 AbsorptionLodInput . . . . .	3
1.3 AbsorptionRendererLodInputData . . . . .	5
1.4 AbsorptionTextureLodInputData . . . . .	7
1.5 AlbedoLod . . . . .	7
1.6 AlbedoLodInput . . . . .	9
1.7 AlbedoRendererLodInputData . . . . .	11
1.8 AnimatedWavesLod . . . . .	12
1.9 AnimatedWavesLodInput . . . . .	16
1.10 AnimatedWavesRendererLodInputData . . . . .	20
1.11 ClipLod . . . . .	21
1.12 ClipLodInput . . . . .	23
1.13 ClipRendererLodInputData . . . . .	26
1.14 ClipTextureLodInputData . . . . .	27
1.15 CollisionLayer . . . . .	28
1.16 CollisionLayers . . . . .	28
1.17 CollisionSource . . . . .	29
1.18 ColorLod . . . . .	30
1.19 Control . . . . .	32
1.20 Controller . . . . .	33
1.21 CustomTimeProvider . . . . .	34
1.22 CutsceneTimeProvider . . . . .	35
1.23 DefaultClippingState . . . . .	37
1.24 DepthGeometryLodInputData . . . . .	37
1.25 DepthLod . . . . .	38
1.26 DepthLodInput . . . . .	40
1.27 DepthProbe . . . . .	43
1.28 DepthProbeMode . . . . .	46
1.29 DepthProbeRefreshMode . . . . .	46
1.30 DepthRendererLodInputData . . . . .	47
1.31 DepthTextureLodInputData . . . . .	48
1.32 DirectionalTextureLodInputData . . . . .	49
1.33 DisplacementPass . . . . .	49
1.34 DynamicWavesLod . . . . .	50
1.35 DynamicWavesLodInput . . . . .	52
1.36 DynamicWavesLodSettings . . . . .	55
1.37 DynamicWavesRendererLodInputData . . . . .	56
1.38 FixedControl . . . . .	57
1.39 FloatingObject . . . . .	58

1.40	FloatingObjectModel . . . . .	61
1.41	FloatingObjectProbe . . . . .	61
1.42	FlowLod . . . . .	62
1.43	FlowLodInput . . . . .	64
1.44	FlowRendererLodInputData . . . . .	66
1.45	FlowTextureLodInputData . . . . .	68
1.46	FoamLod . . . . .	68
1.47	FoamLodInput . . . . .	71
1.48	FoamLodSettings . . . . .	73
1.49	FoamRendererLodInputData . . . . .	75
1.50	FoamTextureLodInputData . . . . .	76
1.51	GeometryLodInputData . . . . .	77
1.52	ICollisionProvider . . . . .	77
1.53	IDepthProvider . . . . .	79
1.54	IFlowProvider . . . . .	80
1.55	IQueryProvider . . . . .	81
1.56	ITimeProvider . . . . .	82
1.57	LevelGeometryLodInputData . . . . .	82
1.58	LevelLod . . . . .	83
1.59	LevelLodInput . . . . .	84
1.60	LevelRendererLodInputData . . . . .	87
1.61	LevelTextureLodInputData . . . . .	88
1.62	Lod . . . . .	89
1.63	Lod<T> . . . . .	90
1.64	LodInput . . . . .	93
1.65	LodInputBlend . . . . .	95
1.66	LodInputData . . . . .	96
1.67	LodInputModule . . . . .	96
1.68	LodInputPrimitive . . . . .	98
1.69	LodQuerySource . . . . .	98
1.70	LodSettings . . . . .	99
1.71	LodTextureFormatMode . . . . .	99
1.72	Meniscus . . . . .	100
1.73	NetworkedTimeProvider . . . . .	101
1.74	PersistentLod . . . . .	102
1.75	Placement . . . . .	104
1.76	PortalMode . . . . .	105
1.77	PortalRenderer . . . . .	105
1.78	QualitySettingsOverride . . . . .	106
1.79	QueryEvents . . . . .	108
1.80	QuerySource . . . . .	111
1.81	RayCastHelper . . . . .	111
1.82	RendererLodInputData . . . . .	112
1.83	SampleCollisionHelper . . . . .	114
1.84	SampleDepthHelper . . . . .	120
1.85	SampleFlowHelper . . . . .	120
1.86	ScatteringLod . . . . .	121
1.87	ScatteringLodInput . . . . .	123
1.88	ScatteringRendererLodInputData . . . . .	125
1.89	ScatteringTextureLodInputData . . . . .	127
1.90	ShadowLod . . . . .	127
1.91	ShadowLodInput . . . . .	130
1.92	ShadowRendererLodInputData . . . . .	132
1.93	ShapeFFT . . . . .	133

1.94	ShapeGerstner	139
1.95	ShapeWaves	145
1.96	ShapeWavesRendererLodInputData	150
1.97	ShapeWavesTextureLodInputData	151
1.98	ShorelineVolumeColorSource	152
1.99	SphereWaterInteraction	152
1.100	SurfaceRenderer	155
1.101	TextureLodInputData	157
1.102	TimeProvider	158
1.103	UnderwaterRenderer	158
1.104	WaterBody	161
1.105	WaterCamera	162
1.106	WaterCameraExclusion	162
1.107	WaterDataBackgroundMode	163
1.108	WaterInjectionPoint	164
1.109	WaterReflectionSide	164
1.110	WaterReflections	165
1.111	WaterRenderer	169
1.112	WatertightHull	182
1.113	WatertightHullMode	184
1.114	WaveSampling	184
1.115	WaveSpectrum	185

## 1.1 AbsorptionLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest  
Extends [ColorLod](#)

Overrides the absorption color.

### 1.1.1 Inherited Properties

#### ShorelineColor

Color of the shoreline color.

#### Declaration

```
public Color ShorelineColor { get; set; }
```

#### ShorelineColorFalloff

Shoreline color falloff value.

#### Declaration

```
public float ShorelineColorFalloff { get; set; }
```

#### ShorelineColorMaximumDistance

The maximum distance of the shoreline color.

If using Depth, then it is maximum depth.

#### Declaration

```
public float ShorelineColorMaximumDistance { get; set; }
```

## ShorelineColorSource

Source of the shoreline color.

### Declaration

```
public ShorelineVolumeColorSource ShorelineColorSource { get; set; }
```

## Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

### Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

### Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

### Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

## Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

## Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

## Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

## Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

# 1.2 AbsorptionLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [AbsorptionLod](#).

Attach this to objects that you want to influence the scattering color.

## 1.2.1 Inherited Properties

### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

## Declaration

```
public LodInputBlend Blend { get; set; }
```

## FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

## Declaration

```
public float FeatherWidth { get; set; }
```

## FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

## Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

## Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

## Declaration

```
public LodInputModule Mode { get; }
```

## Queue

The order this input will render.

Order is Queue plus SiblingIndex

## Declaration

```
public int Queue { get; set; }
```

## Weight

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.2.2 Inherited Methods****GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

**1.3 AbsorptionRendererLodInputData**

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

### 1.3.1 Inherited Properties

#### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

#### Declaration

```
public bool CheckShaderName { get; set; }
```

#### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

#### Declaration

```
public bool CheckShaderPasses { get; set; }
```

#### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

#### Declaration

```
public bool DisableRenderer { get; set; }
```

#### OverrideShaderPass

Whether to set the shader pass manually.

#### Declaration

```
public bool OverrideShaderPass { get; set; }
```

#### Renderer

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

#### Declaration

```
public Renderer Renderer { get; set; }
```

#### ShaderPassIndex

The shader pass to execute.

Set to -1 to execute all passes.

## Declaration

```
public int ShaderPassIndex { get; set; }
```

## 1.4 AbsorptionTextureLodInputData

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [TextureLodInputData](#)

Data storage for for the Texture input mode.

### 1.4.1 Inherited Properties

#### Multiplier

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

#### Declaration

```
public Vector4 Multiplier { get; set; }
```

#### Texture

Texture to render into the simulation.

#### Declaration

```
public Texture Texture { get; set; }
```

## 1.5 AlbedoLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [Lod](#)

A color layer that can be composited onto the water surface.

### 1.5.1 Inherited Properties

#### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

## Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

## Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

## Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

## Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

## Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

## Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

### Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.6 AlbedoLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [AlbedoLod](#).

Attach this to objects that you want to influence the surface color.

### 1.6.1 Inherited Properties

#### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

### Declaration

```
public LodInputBlend Blend { get; set; }
```

#### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

### Declaration

```
public float FeatherWidth { get; set; }
```

## FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

## Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputMode) to set the mode via scripting. The mode cannot be changed after creation.

### Declaration

```
public LodInputMode Mode { get; }
```

## Queue

The order this input will render.

Order is Queue plus SiblingIndex

### Declaration

```
public int Queue { get; set; }
```

## Weight

Scales the input.

### Declaration

```
public float Weight { get; set; }
```

## ForceRenderingOff

Disables rendering of input into data, but continues most scripting activities.

### Declaration

```
public bool ForceRenderingOff { get; set; }
```

## Data

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *Shape-WavesTextureLodInputData* ).

## Declaration

```
public LodInputData Data { get; }
```

### 1.6.2 Inherited Methods

#### GetData

##### GetData<T>()

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

## Declaration

```
public T GetData<T>() where T : LodInputData
```

## Return

The casted data.

## 1.7 AlbedoRendererLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

### 1.7.1 Inherited Properties

#### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

## Declaration

```
public bool CheckShaderName { get; set; }
```

**CheckShaderPasses**

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

**Declaration**

```
public bool CheckShaderPasses { get; set; }
```

**DisableRenderer**

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

**Declaration**

```
public bool DisableRenderer { get; set; }
```

**OverrideShaderPass**

Whether to set the shader pass manually.

**Declaration**

```
public bool OverrideShaderPass { get; set; }
```

**Renderer**

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

**Declaration**

```
public Renderer Renderer { get; set; }
```

**ShaderPassIndex**

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.8 AnimatedWavesLod

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *Lod<T>*

Captures waves/shape that is drawn kinematically - there is no frame-to-frame state.

- A combine pass is done which combines downwards from low detail LODs into the high detail LODs.
- The LOD data is passed to the water material when the surface is drawn. *DynamicWavesLod* adds its results into this data. They piggy back off the combine pass and subsequent assignment to the water material.

The RGB channels are the XYZ displacement from a rest plane at water level to the corresponding displaced position on the surface.

## 1.8.1 Properties

### AttenuationInShallows

How much waves are dampened in shallow water.

#### Declaration

```
public float AttenuationInShallows { get; set; }
```

### CollisionLayers

Collision layers to enable.

Some layers will have overhead with CPU, GPU and memory.

#### Declaration

```
public CollisionLayers CollisionLayers { get; set; }
```

### CollisionSource

#### Obsolete

Please use QuerySource instead.

Where to obtain water shape on CPU for physics / gameplay.

#### Declaration

```
public CollisionSource CollisionSource { get; }
```

### ShallowsMaximumDepth

Any water deeper than this will receive full wave strength.

The lower the value, the less effective the depth cache will be at attenuating very large waves. Set to the maximum value (1,000) to disable.

#### Declaration

```
public float ShallowsMaximumDepth { get; set; }
```

## WaveResolutionMultiplier

Shifts wavelengths to maintain quality for higher resolutions.

Set this to 2 to improve wave quality. In some cases like flowing rivers, this can make a substantial difference to visual stability. We recommend doubling the Resolution on the WaterRenderer component to preserve detail after making this change.

### Declaration

```
public float WaveResolutionMultiplier { get; set; }
```

## WaveSampling

The wave sampling method to determine quality and performance.

### Declaration

```
public WaveSampling WaveSampling { get; set; }
```

## 1.8.2 Inherited Properties

### MaximumQueryCount

Maximum number of queries that can be performed when using GPU queries.

### Declaration

```
public int MaximumQueryCount { get; }
```

### QuerySource

Where to obtain water data on CPU for physics / gameplay.

### Declaration

```
public LodQuerySource QuerySource { get; }
```

### Provider

Provides data from the GPU to CPU.

### Declaration

```
public T Provider { get; set; }
```

### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

## Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

## Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

## Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

## Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

## Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

## Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

**TextureFormatMode**

Chooses a texture format based on a preset value.

**Declaration**

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

**CompatibleTextureFormat**

This is the platform compatible texture format that will used.

**Declaration**

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.9 AnimatedWavesLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [AnimatedWavesLod](#).

Attach this to objects that you want to render into the displacement textures to affect the water shape.

### 1.9.1 Properties

**DisplacementPass**

When to render the input into the displacement data.

**Declaration**

```
public DisplacementPass DisplacementPass { get; set; }
```

**FilterByWavelength**

Whether to filter this input by wavelength.

If disabled, it will render to all LODs.

**Declaration**

```
public bool FilterByWavelength { get; set; }
```

**MaximumDisplacementHorizontal**

Inform the system how much this input will displace the water surface horizontally.

This is used to set bounding box widths for the water chunks.

**Declaration**

```
public float MaximumDisplacementHorizontal { get; set; }
```

**MaximumDisplacementVertical**

Inform the system how much this input will displace the water surface vertically.

This is used to set bounding box heights for the water chunks.

**Declaration**

```
public float MaximumDisplacementVertical { get; set; }
```

**OctaveWavelength**

Which octave to render into.

For example, set this to 2 to render into the 2m-4m octave. These refer to the same octaves as the wave spectrum editor.

**Declaration**

```
public float OctaveWavelength { get; set; }
```

**RenderPostCombine****Obsolete**

Please use DisplacementPass instead.

When to render the input into the displacement data.

If enabled, it renders into all LODs of the simulation after the combine step rather than before with filtering. Furthermore, it will also affect dynamic waves.

**Declaration**

```
public bool RenderPostCombine { get; set; }
```

**ReportRendererBounds**

Use the bounding box of an attached renderer component to determine the maximum vertical displacement.

**Declaration**

```
public bool ReportRendererBounds { get; set; }
```

**1.9.2 Inherited Properties****Blend**

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

### Declaration

```
public LodInputBlend Blend { get; set; }
```

### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

### Declaration

```
public float FeatherWidth { get; set; }
```

### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

### Declaration

```
public LodInputModule Mode { get; }
```

### Queue

The order this input will render.

Order is Queue plus SiblingIndex

### Declaration

```
public int Queue { get; set; }
```

## Weight

Scales the input.

### Declaration

```
public float Weight { get; set; }
```

## ForceRenderingOff

Disables rendering of input into data, but continues most scripting activities.

### Declaration

```
public bool ForceRenderingOff { get; set; }
```

## Data

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

### Declaration

```
public LodInputData Data { get; }
```

## 1.9.3 Inherited Methods

### GetData

#### GetData<T>()

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

### Declaration

```
public T GetData<T>() where T : LodInputData
```

### Return

The casted data.

## 1.10 AnimatedWavesRendererLodInputData

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [RendererLodInputData](#)

Data storage for for the Renderer input mode.

### 1.10.1 Inherited Properties

#### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

#### Declaration

```
public bool CheckShaderName { get; set; }
```

#### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

#### Declaration

```
public bool CheckShaderPasses { get; set; }
```

#### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

#### Declaration

```
public bool DisableRenderer { get; set; }
```

#### OverrideShaderPass

Whether to set the shader pass manually.

#### Declaration

```
public bool OverrideShaderPass { get; set; }
```

#### Renderer

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

**Declaration**

```
public Renderer Renderer { get; set; }
```

**ShaderPassIndex**

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.11 ClipLod

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *Lod*

Drives water surface clipping (carving holes).

0-1 values, surface clipped when > 0.5.

### 1.11.1 Properties

**DefaultClippingState**

The default clipping behaviour.

Whether to clip nothing by default (and clip inputs remove patches of surface), or to clip everything by default (and clip inputs add patches of surface).

**Declaration**

```
public DefaultClippingState DefaultClippingState { get; set; }
```

### 1.11.2 Inherited Properties

**Blur**

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

**Declaration**

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

### Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

### Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

### Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overridden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

**Declaration**

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

**CompatibleTextureFormat**

This is the platform compatible texture format that will used.

**Declaration**

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.12 ClipLodInput

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *LodInput*

Registers a custom input to the *ClipLod*.

Attach this to objects that you want to use to clip the surface of the water.

### 1.12.1 Properties

**Inverted**

Removes clip surface data instead of adding it.

**Declaration**

```
public bool Inverted { get; set; }
```

**Primitive**

The primitive to render (signed distance) into the simulation.

**Declaration**

```
public LodInputPrimitive Primitive { get; set; }
```

**WaterHeightDistanceCulling**

Prevents inputs from cancelling each other out when aligned vertically.

It is imperfect so custom logic might be needed for your use case.

**Declaration**

```
public bool WaterHeightDistanceCulling { get; set; }
```

## 1.12.2 Inherited Properties

### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

### Declaration

```
public LodInputBlend Blend { get; set; }
```

### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

### Declaration

```
public float FeatherWidth { get; set; }
```

### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

### Declaration

```
public LodInputModule Mode { get; }
```

### Queue

The order this input will render.

Order is Queue plus SiblingIndex

**Declaration**

```
public int Queue { get; set; }
```

**Weight**

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *Shape-WavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.12.3 Inherited Methods****GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

## Return

The casted data.

## 1.13 ClipRendererLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

### 1.13.1 Inherited Properties

#### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

#### Declaration

```
public bool CheckShaderName { get; set; }
```

#### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

#### Declaration

```
public bool CheckShaderPasses { get; set; }
```

#### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

#### Declaration

```
public bool DisableRenderer { get; set; }
```

#### OverrideShaderPass

Whether to set the shader pass manually.

#### Declaration

```
public bool OverrideShaderPass { get; set; }
```

**Renderer**

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

**Declaration**

```
public Renderer Renderer { get; set; }
```

**ShaderPassIndex**

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.14 ClipTextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *TextureLodInputData*

Data storage for for the Texture input mode.

### 1.14.1 Inherited Properties

**Multiplier**

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

**Texture**

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.15 CollisionLayer

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

A layer/event where queries are executed.

### 1.15.1 Properties

#### Everything

Include all displacement.

#### Declaration

```
Everything = 0
```

#### AfterAnimatedWaves

Only include Animated Waves.

#### Declaration

```
AfterAnimatedWaves = 1
```

#### AfterDynamicWaves

Include Animated Waves and Dynamic Waves.

#### Declaration

```
AfterDynamicWaves = 2
```

## 1.16 CollisionLayers

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Flags to enable extra collision layers.

### 1.16.1 Properties

#### Nothing

No extra layers (ie single layer).

#### Declaration

```
Nothing = 0
```

## DynamicWaves

Separate layer for dynamic waves.

Dynamic waves are normally combined together for efficiency. By enabling this layer, dynamic waves are combined and added in a separate pass.

### Declaration

```
DynamicWaves = 2
```

## Displacement

Extra displacement layer for visual displacement.

### Declaration

```
Displacement = 4
```

## Everything

All layers.

### Declaration

```
Everything = -1
```

## 1.17 CollisionSource

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

### Obsolete

Please use QuerySource and LodQuerySource.

The source of collisions (ie water shape).

### 1.17.1 Properties

#### None

No collision source. Flat water.

### Declaration

```
None = 0
```

**GPU**

Uses AsyncGPUReadback to retrieve data from GPU to CPU.

This is the most optimal approach.

**Declaration**

```
GPU = 2
```

**CPU**

Computes data entirely on the CPU.

**Declaration**

```
CPU = 3
```

## 1.18 ColorLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [Lod](#)

Contains shared functionality for [AbsorptionLod](#) and [ScatteringLod](#).

### 1.18.1 Properties

**ShorelineColor**

Color of the shoreline color.

**Declaration**

```
public Color ShorelineColor { get; set; }
```

**ShorelineColorFalloff**

Shoreline color falloff value.

**Declaration**

```
public float ShorelineColorFalloff { get; set; }
```

**ShorelineColorMaximumDistance**

The maximum distance of the shoreline color.

If using Depth, then it is maximum depth.

## Declaration

```
public float ShorelineColorMaximumDistance { get; set; }
```

## ShorelineColorSource

Source of the shoreline color.

## Declaration

```
public ShorelineVolumeColorSource ShorelineColorSource { get; set; }
```

## 1.18.2 Inherited Properties

### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

## Declaration

```
public bool Blur { get; set; }
```

### BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

## Declaration

```
public int BlurIterations { get; set; }
```

### Enabled

Whether the simulation is enabled.

## Declaration

```
public bool Enabled { get; set; }
```

### OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

## Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

## Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

## Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

## Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

## Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

# 1.19 Control

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest.Watercraft

Controls provide input whether from the player or otherwise. Extend to implement a control. See derived classes for examples.

## 1.19.1 Properties

### Input

Provides input for controllers. XYZ is steer, float and drive respectively.

### Declaration

```
public abstract Vector3 Input { get; }
```

## 1.20 Controller

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest.Watercraft

A simple watercraft controller.

### 1.20.1 Properties

#### BuoyancyCurveFactor

Applies a curve to buoyancy changes.

### Declaration

```
public AnimationCurve BuoyancyCurveFactor { get; set; }
```

#### Control

The accompanied control script to take input from.

### Declaration

```
public Control Control { get; set; }
```

#### FloatingObject

The accompanied buoyancy script.

### Declaration

```
public FloatingObject FloatingObject { get; set; }
```

#### ForceHeightOffset

Vertical offset from the center of mass for where move force should be applied.

### Declaration

```
public float ForceHeightOffset { get; set; }
```

## SteerPower

How quickly the watercraft turns from steering.

### Declaration

```
public float SteerPower { get; set; }
```

## ThrustPower

How quickly the watercraft moves from thrust.

### Declaration

```
public float ThrustPower { get; set; }
```

## TurningHeel

Rolls the watercraft when turning.

### Declaration

```
public float TurningHeel { get; set; }
```

## 1.21 CustomTimeProvider

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *TimeProvider*, *ITimeProvider*

This time provider fixes the water time at a custom value which is usable for testing/debugging.

### 1.21.1 Properties

#### DeltaTime

The delta time override value.

### Declaration

```
public float DeltaTime { get; set; }
```

#### OverrideDeltaTime

Whether to override the water simulation time.

This in particular affects dynamic elements of the simulation like the foam simulation and the ripple simulation.

## Declaration

```
public bool OverrideDeltaTime { get; set; }
```

## OverrideTime

Whether to override the water simulation time.

## Declaration

```
public bool OverrideTime { get; set; }
```

## Paused

Freeze progression of time. Only works properly in Play mode.

## Declaration

```
public bool Paused { get; set; }
```

## TimeOverride

The time override value.

## Declaration

```
public float TimeOverride { get; set; }
```

## Time

Current time.

## Declaration

```
public override float Time { get; }
```

## Delta

Delta time.

## Declaration

```
public override float Delta { get; }
```

# 1.22 CutsceneTimeProvider

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *TimeProvider*, *ITimeProvider*

This time provider feeds a Timeline time to the water system, using a Playable Director.

### 1.22.1 Properties

#### AssignToWaterComponentOnEnable

Assign this time provider to the water system when this component becomes active.

#### Declaration

```
public bool AssignToWaterComponentOnEnable { get; set; }
```

#### PlayableDirector

Playable Director to take time from.

#### Declaration

```
public PlayableDirector PlayableDirector { get; set; }
```

#### RestorePreviousTimeProviderOnDisable

Restore the time provider that was previously assigned to water system when this component disables.

#### Declaration

```
public bool RestorePreviousTimeProviderOnDisable { get; set; }
```

#### TimeOffset

Time offset which will be added to the Timeline time.

#### Declaration

```
public float TimeOffset { get; set; }
```

#### Time

Current time.

If there is a PlayableDirector which is playing, return its time, otherwise use the *TimeProvider* being used before this component initialised, else fallback to a default *TimeProvider*.

#### Declaration

```
public override float Time { get; }
```

#### Delta

Delta time.

**Declaration**

```
public override float Delta { get; }
```

## 1.23 DefaultClippingState

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest

The default state for clipping.

### 1.23.1 Properties

**NothingClipped**

By default, nothing is clipped. Use clip inputs to remove water.

**Declaration**

```
NothingClipped = 0
```

**EverythingClipped**

By default, everything is clipped. Use clip inputs to add water.

**Declaration**

```
EverythingClipped = 1
```

## 1.24 DepthGeometryLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *GeometryLodInputData*

Data storage for for the Geometry input mode.

### 1.24.1 Inherited Properties

**Geometry**

Geometry to render into the simulation.

**Declaration**

```
public Mesh Geometry { get; set; }
```

## 1.25 DepthLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [Lod<T>](#)

Data that gives depth of the water (height of sea level above water floor).

### 1.25.1 Properties

#### EnableSignedDistanceFields

Support signed distance field data generated from the depth probes.

Requires a two component texture format.

#### Declaration

```
public bool EnableSignedDistanceFields { get; set; }
```

#### IncludeTerrainHeight

Whether to include the terrain height automatically.

This will not include terrain details, nor will it produce a signed-distance field. There may also be a slight deviation due to differences in height data and terrain mesh. In these cases, please use the DepthProbe.

#### Declaration

```
public bool IncludeTerrainHeight { get; set; }
```

### 1.25.2 Inherited Properties

#### MaximumQueryCount

Maximum number of queries that can be performed when using GPU queries.

#### Declaration

```
public int MaximumQueryCount { get; }
```

#### QuerySource

Where to obtain water data on CPU for physics / gameplay.

#### Declaration

```
public LodQuerySource QuerySource { get; }
```

## Provider

Provides data from the GPU to CPU.

### Declaration

```
public T Provider { get; set; }
```

## Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

### Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

### Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

### Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

**Declaration**

```
public int Resolution { get; set; }
```

**TextureFormat**

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

**Declaration**

```
public GraphicsFormat TextureFormat { get; set; }
```

**TextureFormatMode**

Chooses a texture format based on a preset value.

**Declaration**

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

**CompatibleTextureFormat**

This is the platform compatible texture format that will used.

**Declaration**

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.26 DepthLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [DepthLod](#).

Attach this to objects that you want to use to add water depth.

Renders depth every frame and should only be used for dynamic objects. For static objects, use a [DepthProbe](#)

### 1.26.1 Properties

**CopySignedDistanceField**

Whether to copy the signed distance field.

**Declaration**

```
public bool CopySignedDistanceField { get; set; }
```

## Relative

Whether the data is relative to the input height.

Useful for procedural placement.

### Declaration

```
public bool Relative { get; set; }
```

## 1.26.2 Inherited Properties

### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

### Declaration

```
public LodInputBlend Blend { get; set; }
```

### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

### Declaration

```
public float FeatherWidth { get; set; }
```

### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

**Declaration**

```
public LodInputMode Mode { get; }
```

**Queue**

The order this input will render.

Order is Queue plus SiblingIndex

**Declaration**

```
public int Queue { get; set; }
```

**Weight**

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *Shape-WavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.26.3 Inherited Methods****GetData**

**GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

## 1.27 DepthProbe

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Captures scene height / water depth, and renders it into the simulation.

Caches the operation to avoid rendering it every frame. This should be used for static geometry, dynamic objects should be tagged with the *DepthLodInput* component.

### 1.27.1 Properties

**AdditionalJumpFloodRounds**

How many additional Jump Flood rounds to use.

The standard number of rounds is  $\log_2(\text{resolution})$ . Additional rounds can reduce innaccuracies.

**Declaration**

```
public int AdditionalJumpFloodRounds { get; set; }
```

**CaptureRange**

The far and near plane of the depth probe camera respectively, relative to the transform.

Depth is captured top-down and orthographically. The gizmo will visualize this range as the bottom box.

**Declaration**

```
public Vector2 CaptureRange { get; set; }
```

**EnableBackFaceInclusion**

Increase coverage by testing mesh back faces within the Fill Holes area.

Uses the back-faces to include meshes where the front-face is within the Fill Holes area and the back-face is within the capture area. An example would be an upright cylinder not over a hole but was not captured due to the top being clipped by the near plane.

## Declaration

```
public bool EnableBackFaceInclusion { get; set; }
```

## FillHolesCaptureHeight

Fills holes left by the maximum of the capture range.

Setting the maximum capture range lower than the highest point of geometry can be useful for eliminating depth artifacts from overhangs, but the side effect is there will be a hole in the depth data where geometry is clipped by the near plane. This will only capture where the holes are to fill them in. This height is relative to the maximum capture range. Set to zero to skip.

## Declaration

```
public float FillHolesCaptureHeight { get; set; }
```

## GenerateSignedDistanceField

Generate a signed distance field for the shoreline.

## Declaration

```
public bool GenerateSignedDistanceField { get; set; }
```

## Layers

The layers to render into the probe.

## Declaration

```
public LayerMask Layers { get; set; }
```

## QualitySettingsOverride

Overrides global quality settings.

## Declaration

```
public QualitySettingsOverride QualitySettingsOverride { get; }
```

## RefreshMode

Controls how the probe is refreshed in the Player.

Call Populate() if scripting.

## Declaration

```
public DepthProbeRefreshMode RefreshMode { get; set; }
```

## Resolution

The resolution of the probe.

Lower will be more efficient.

## Declaration

```
public int Resolution { get; set; }
```

## SavedTexture

Baked probe.

Can only bake in edit mode.

## Declaration

```
public Texture2D SavedTexture { get; set; }
```

## Type

Specifies the setup for this probe.

## Declaration

```
public DepthProbeMode Type { get; set; }
```

## 1.27.2 Methods

### Populate

#### Populate()

Populates the *DepthProbe* (including re-baking).

Call this method if using DepthProbeRefreshMode.ViaScripting , or if needing the probe to be updated re-baked (re-baking editor only).

## Declaration

```
public void Populate()
```

## 1.27.3 Events

### OnBeforeRender

Invoked before the *DepthProbe* camera renders.

## Declaration

```
public static Action<DepthProbe> OnBeforeRender { get; set; }
```

## OnAfterRender

Invoked after the *DepthProbe* camera renders.

### Declaration

```
public static Action<DepthProbe> OnAfterRender { get; set; }
```

## 1.28 DepthProbeMode

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest

*DepthProbe* ‘s update mode.

### 1.28.1 Properties

#### RealTime

Update in real-time in accordance to refresh mode.

#### Declaration

```
RealTime = 0
```

#### Baked

Baked in the editor.

#### Declaration

```
Baked = 1
```

## 1.29 DepthProbeRefreshMode

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest

How the *DepthProbe* refreshes when using DepthProbeMode.RealTime.

### 1.29.1 Properties

#### OnStart

Populates the DepthProbe in Start.

#### Declaration

```
OnStart = 0
```

## ViaScripting

Requires manual updating via DepthProbe.Populate.

## Declaration

```
ViaScripting = 2
```

# 1.30 DepthRendererLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

## 1.30.1 Inherited Properties

### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

## Declaration

```
public bool CheckShaderName { get; set; }
```

### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

## Declaration

```
public bool CheckShaderPasses { get; set; }
```

### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

## Declaration

```
public bool DisableRenderer { get; set; }
```

### OverrideShaderPass

Whether to set the shader pass manually.

**Declaration**

```
public bool OverrideShaderPass { get; set; }
```

**Renderer**

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

**Declaration**

```
public Renderer Renderer { get; set; }
```

**ShaderPassIndex**

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.31 DepthTextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *TextureLodInputData*

Data storage for for the Texture input mode.

### 1.31.1 Inherited Properties

**Multiplier**

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

**Texture**

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.32 DirectionalTextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *TextureLodInputData*

Data storage for for the Texture input mode.

### 1.32.1 Properties

#### NegativeValues

Whether the texture supports negative values.

**Declaration**

```
public bool NegativeValues { get; set; }
```

### 1.32.2 Inherited Properties

#### Multiplier

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

#### Texture

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.33 DisplacementPass

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest

The pass to render displacement into.

### 1.33.1 Properties

#### LodDependent

Displacement that is dependent on an LOD (eg waves).

Uses filtering to determine which LOD to write to.

#### Declaration

```
LodDependent = 0
```

#### LodIndependent

Renders to all LODs.

#### Declaration

```
LodIndependent = 1
```

#### LodIndependentLast

Renders to all LODs, but as a separate pass.

Typically used to render visual displacement which does not affect collisions.

#### Declaration

```
LodIndependentLast = 2
```

## 1.34 DynamicWavesLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [PersistentLod](#)

A dynamic shape simulation that moves around with a displacement LOD.

### 1.34.1 Properties

#### AttenuationInShallows

How much waves are damped in shallow water.

#### Declaration

```
public float AttenuationInShallows { get; set; }
```

#### Settings

Settings for fine tuning this simulation.

## Declaration

```
public DynamicWavesLodSettings Settings { get; set; }
```

### 1.34.2 Inherited Properties

#### SimulationFrequency

Frequency to run the simulation, in updates per second.

Lower frequencies are more efficient but may lead to visible jitter or slowness.

#### Declaration

```
public int SimulationFrequency { get; set; }
```

#### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

#### Declaration

```
public bool Blur { get; set; }
```

#### BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

#### Declaration

```
public int BlurIterations { get; set; }
```

#### Enabled

Whether the simulation is enabled.

#### Declaration

```
public bool Enabled { get; set; }
```

#### OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

**Declaration**

```
public bool OverrideResolution { get; set; }
```

**Resolution**

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

**Declaration**

```
public int Resolution { get; set; }
```

**TextureFormat**

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

**Declaration**

```
public GraphicsFormat TextureFormat { get; set; }
```

**TextureFormatMode**

Chooses a texture format based on a preset value.

**Declaration**

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

**CompatibleTextureFormat**

This is the platform compatible texture format that will used.

**Declaration**

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.35 DynamicWavesLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [DynamicWavesLod](#).

Attach this to objects that you want to influence the simulation, such as ripples etc.

### 1.35.1 Inherited Properties

#### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

#### Declaration

```
public LodInputBlend Blend { get; set; }
```

#### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

#### Declaration

```
public float FeatherWidth { get; set; }
```

#### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

#### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

#### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

#### Declaration

```
public LodInputModule Mode { get; }
```

#### Queue

The order this input will render.

Order is Queue plus SiblingIndex

**Declaration**

```
public int Queue { get; set; }
```

**Weight**

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *Shape-WavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.35.2 Inherited Methods****GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

## 1.36 DynamicWavesLodSettings

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodSettings](#)

Settings for fine tuning the *DynamicWavesLod*.

### 1.36.1 Properties

#### CourantNumber

Stability control.

Lower values means more stable simulation, but may slow down some dynamic waves. This value should be set as large as possible until simulation instabilities/flickering begin to appear. Default is 0.7.

#### Declaration

```
public float CourantNumber { get; set; }
```

#### Damping

How much energy is dissipated each frame.

Helps simulation stability, but limits how far ripples will propagate. Set this as large as possible/acceptable. Default is 0.05.

#### Declaration

```
public float Damping { get; set; }
```

#### DisplaceClamp

Clamp displacement to help prevent self-intersection in steep waves.

Zero means unclamped.

#### Declaration

```
public float DisplaceClamp { get; set; }
```

#### GravityMultiplier

Multiplier for gravity.

More gravity means dynamic waves will travel faster. Higher values can be a source of instability.

**Declaration**

```
public float GravityMultiplier { get; set; }
```

**HorizontalDisplace**

Induce horizontal displacements to sharpen simulated waves.

**Declaration**

```
public float HorizontalDisplace { get; set; }
```

## 1.37 DynamicWavesRendererLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

### 1.37.1 Inherited Properties

**CheckShaderName**

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

**Declaration**

```
public bool CheckShaderName { get; set; }
```

**CheckShaderPasses**

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

**Declaration**

```
public bool CheckShaderPasses { get; set; }
```

**DisableRenderer**

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

**Declaration**

```
public bool DisableRenderer { get; set; }
```

### OverrideShaderPass

Whether to set the shader pass manually.

#### Declaration

```
public bool OverrideShaderPass { get; set; }
```

### Renderer

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

#### Declaration

```
public Renderer Renderer { get; set; }
```

### ShaderPassIndex

The shader pass to execute.

Set to -1 to execute all passes.

#### Declaration

```
public int ShaderPassIndex { get; set; }
```

## 1.38 FixedControl

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest.Watercraft

Extends *Control*

Constantly moves/turns.

### 1.38.1 Properties

#### Move

Constantly move.

#### Declaration

```
public float Move { get; set; }
```

#### Turn

Constantly turn.

**Declaration**

```
public float Turn { get; set; }
```

**Input**

Provides input for controllers. XYZ is steer, float and drive respectively.

**Declaration**

```
public override Vector3 Input { get; }
```

## 1.39 FloatingObject

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Physics including buoyancy and drag.

### 1.39.1 Properties

**AccelerateDownhill**

Approximate hydrodynamics of ‘surfing’ down waves.

**Declaration**

```
public float AccelerateDownhill { get; set; }
```

**AngularDrag**

Angular drag when in water.

Additive to the angular drag declared on the rigid body.

**Declaration**

```
public float AngularDrag { get; set; }
```

**BuoyancyForceStrength**

Strength of buoyancy force.

For probes, roughly a mass to force ratio of 100 to 1 to keep the center of mass near the surface. For Align Normal, default value is for a default sphere with a default rigidbody.

**Declaration**

```
public float BuoyancyForceStrength { get; set; }
```

## BuoyancyTorqueStrength

Strength of torque applied to match boat orientation to water normal.

### Declaration

```
public float BuoyancyTorqueStrength { get; set; }
```

## CenterToBottomOffset

Height offset from transform center to bottom of boat (if any).

Default value is for a default sphere. Having this value be an accurate measurement from center to bottom is not necessary.

### Declaration

```
public float CenterToBottomOffset { get; set; }
```

## Drag

Drag when in water.

Additive to the drag declared on the rigid body.

### Declaration

```
public Vector3 Drag { get; set; }
```

## ForceHeightOffset

Vertical offset for where drag force should be applied.

### Declaration

```
public float ForceHeightOffset { get; set; }
```

## Layer

Which water collision layer to target.

### Declaration

```
public CollisionLayer Layer { get; set; }
```

## MaximumBuoyancyForce

Clamps the buoyancy force to this value.

Useful for handling fully submerged objects.

## Declaration

```
public float MaximumBuoyancyForce { get; set; }
```

## Model

The model to use for buoyancy.

Align Normal is simple and only uses a few queries whilst Probes is more advanced and uses a few queries per probe. Cannot be changed at runtime after Start.

## Declaration

```
public FloatingObjectModel Model { get; set; }
```

## ObjectLength

Length dimension of boat.

Only used if Use Boat Length is enabled.

## Declaration

```
public float ObjectLength { get; set; }
```

## ObjectWidth

Width of object for physics purposes.

The larger this value, the more filtered/smooth the wave response will be. If larger wavelengths cannot be filtered, increase the LOD Levels

## Declaration

```
public float ObjectWidth { get; set; }
```

## Probes

Query points for buoyancy.

Only applicable to Probes model.

## Declaration

```
public FloatingObjectProbe[] Probes { get; set; }
```

## RigidBody

The rigid body to affect.

It will automatically get the sibling rigid body if not set.

**Declaration**

```
public Rigidbody Rigidbody { get; set; }
```

**UseObjectLength**

Computes a separate normal based on boat length to get more accurate orientations.

Requires the cost of an extra collision sample.

**Declaration**

```
public bool UseObjectLength { get; set; }
```

**InWater**

Is any part of this object in water.

**Declaration**

```
public bool InWater { get; }
```

## 1.40 FloatingObjectModel

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Different physics models for *FloatingObject*

### 1.40.1 Properties

**AlignNormal**

A simple model which aligns the object with the wave normal.

**Declaration**

```
AlignNormal = 0
```

**Probes**

A more advanced model which samples water at the probes positions.

**Declaration**

```
Probes = 1
```

## 1.41 FloatingObjectProbe

Struct in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Probes for the *FloatingObject* FloatingObjectModel.Probes model.

### 1.41.1 Properties

#### \_Weight

How much this probe affects the outcome (not a physical weight).

#### Declaration

```
public float _Weight
```

#### \_Position

The position of the probe.

#### Declaration

```
public Vector3 _Position
```

## 1.42 FlowLod

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *Lod<T>*

Simulates horizontal motion of water.

### 1.42.1 Inherited Properties

#### MaximumQueryCount

Maximum number of queries that can be performed when using GPU queries.

#### Declaration

```
public int MaximumQueryCount { get; }
```

#### QuerySource

Where to obtain water data on CPU for physics / gameplay.

#### Declaration

```
public LodQuerySource QuerySource { get; }
```

#### Provider

Provides data from the GPU to CPU.

## Declaration

```
public T Provider { get; set; }
```

## Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

## Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

## Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

## Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

## Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

## Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overridden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will be used.

### Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.43 FlowLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [FlowLod](#).

Attach this to objects that you want to influence the horizontal flow of the water volume.

### 1.43.1 Inherited Properties

#### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

### Declaration

```
public LodInputBlend Blend { get; set; }
```

### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

#### Declaration

```
public float FeatherWidth { get; set; }
```

### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

#### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

#### Declaration

```
public LodInputModule Mode { get; }
```

### Queue

The order this input will render.

Order is Queue plus SiblingIndex

#### Declaration

```
public int Queue { get; set; }
```

### Weight

Scales the input.

#### Declaration

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.43.2 Inherited Methods****GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

**1.44 FlowRendererLodInputData**

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for the Renderer input mode.

### 1.44.1 Inherited Properties

#### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

#### Declaration

```
public bool CheckShaderName { get; set; }
```

#### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

#### Declaration

```
public bool CheckShaderPasses { get; set; }
```

#### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

#### Declaration

```
public bool DisableRenderer { get; set; }
```

#### OverrideShaderPass

Whether to set the shader pass manually.

#### Declaration

```
public bool OverrideShaderPass { get; set; }
```

#### Renderer

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

#### Declaration

```
public Renderer Renderer { get; set; }
```

#### ShaderPassIndex

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.45 FlowTextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest  
Extends *DirectionalTextureLodInputData*

Data storage for for the Texture input mode.

### 1.45.1 Inherited Properties

**NegativeValues**

Whether the texture supports negative values.

**Declaration**

```
public bool NegativeValues { get; set; }
```

**Multiplier**

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

**Texture**

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.46 FoamLod

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest  
Extends *PersistentLod*

A persistent foam simulation that moves around with a displacement LOD. The input is fully combined water surface shape.

## 1.46.1 Properties

### Prewarm

Prewarms the simulation on load and teleports.

Results are only an approximation.

### Declaration

```
public bool Prewarm { get; set; }
```

### Settings

Settings for fine tuning this simulation.

### Declaration

```
public FoamLodSettings Settings { get; set; }
```

## 1.46.2 Inherited Properties

### SimulationFrequency

Frequency to run the simulation, in updates per second.

Lower frequencies are more efficient but may lead to visible jitter or slowness.

### Declaration

```
public int SimulationFrequency { get; set; }
```

### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

### Declaration

```
public bool Blur { get; set; }
```

### BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

### Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

### Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

### Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

## Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.47 FoamLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [FoamLod](#).

Attach this to objects that you want to influence the foam simulation, such as depositing foam on the surface.

### 1.47.1 Inherited Properties

#### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

#### Declaration

```
public LodInputBlend Blend { get; set; }
```

#### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

#### Declaration

```
public float FeatherWidth { get; set; }
```

#### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

#### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

#### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

**Declaration**

```
public LodInputMode Mode { get; }
```

**Queue**

The order this input will render.

Order is Queue plus SiblingIndex

**Declaration**

```
public int Queue { get; set; }
```

**Weight**

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *Shape-WavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.47.2 Inherited Methods****GetData**

**GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

## 1.48 FoamLodSettings

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *LodSettings*

Settings for fine tuning the *FoamLod*.

### 1.48.1 Properties

**FilterWaves**

The minimum LOD to sample waves from.

Zero means all waves and increasing will exclude lower wavelengths which can help with too much foam near the camera.

**Declaration**

```
public int FilterWaves { get; set; }
```

**FoamFadeRate**

How quickly foam dissipates.

Low values mean foam remains on surface for longer. This setting should be balanced with the generation \*strength\* parameters below.

**Declaration**

```
public float FoamFadeRate { get; set; }
```

**Maximum**

Foam will not exceed this value in the simulation which can be used to prevent foam from accumulating too much.

## Declaration

```
public float Maximum { get; set; }
```

### ShorelineFoamMaximumDepth

Foam will be generated in water shallower than this depth.

Controls how wide the band of foam at the shoreline will be. Note that this is not a distance to shoreline, but a threshold on water depth, so the width of the foam band can vary based on terrain slope. To address this limitation we allow foam to be manually added from geometry or from a texture, see the next section.

## Declaration

```
public float ShorelineFoamMaximumDepth { get; set; }
```

### ShorelineFoamPriming

Primes foam when terrain height is this value above water.

This ignores other foam settings and writes a constant foam value.

## Declaration

```
public float ShorelineFoamPriming { get; set; }
```

### ShorelineFoamStrength

Scales intensity of foam generated in shallow water.

This setting should be balanced with the Foam Fade Rate setting.

## Declaration

```
public float ShorelineFoamStrength { get; set; }
```

### WaveFoamCoverage

How much of the waves generate foam.

Higher values will lower the threshold for foam generation, giving a larger area.

## Declaration

```
public float WaveFoamCoverage { get; set; }
```

### WaveFoamStrength

Scales intensity of foam generated from waves.

This setting should be balanced with the Foam Fade Rate setting.

## Declaration

```
public float WaveFoamStrength { get; set; }
```

# 1.49 FoamRendererLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

## 1.49.1 Inherited Properties

### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

### Declaration

```
public bool CheckShaderName { get; set; }
```

### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

### Declaration

```
public bool CheckShaderPasses { get; set; }
```

### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

### Declaration

```
public bool DisableRenderer { get; set; }
```

### OverrideShaderPass

Whether to set the shader pass manually.

### Declaration

```
public bool OverrideShaderPass { get; set; }
```

## Renderer

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

## Declaration

```
public Renderer Renderer { get; set; }
```

## ShaderPassIndex

The shader pass to execute.

Set to -1 to execute all passes.

## Declaration

```
public int ShaderPassIndex { get; set; }
```

# 1.50 FoamTextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *TextureLodInputData*

Data storage for for the Texture input mode.

## 1.50.1 Inherited Properties

### Multiplier

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

## Declaration

```
public Vector4 Multiplier { get; set; }
```

### Texture

Texture to render into the simulation.

## Declaration

```
public Texture Texture { get; set; }
```

## 1.51 GeometryLodInputData

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest  
Extends [LodInputData](#)

Data storage for for the Geometry input mode.

### 1.51.1 Properties

#### Geometry

Geometry to render into the simulation.

#### Declaration

```
public Mesh Geometry { get; set; }
```

## 1.52 ICollisionProvider

Interface in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Interface for an object that returns water surface displacement and height.

### 1.52.1 Methods

#### Query

```
Query(int, float, Vector3[], float[], Vector3[], Vector3[], CollisionLayer, Vector3?)
```

Query water physical data at a set of points. Pass in null to any out parameters that are not required.

#### Declaration

```
int Query(int hash, float minimumLength, Vector3[] points, float[] heights, Vector3[] ↵
    ↵normals, Vector3[] velocities, CollisionLayer layer = CollisionLayer.Everything, ↵
    ↵Vector3? center = null)
```

## Parameters

hash	Unique ID for calling code. Typically acquired by calling GetHashCode.
minimumLength	The minimum spatial length of the object, such as the width of a boat. Useful for filtering out detail when not needed. Set to zero to get full available detail.
points	The world space points that will be queried.
heights	Resulting heights (displacement Y + sea level) at the query positions. Pass null if this information is not required.
normals	Resulting normals at the query positions. Pass null if this information is not required.
velocities	Resulting velocities at the query positions. Pass null if this information is not required.
layer	The layer this query targets.
center	The center of all the query positions. Used to choose the closest query provider.

## Return

The status of the query.

```
Query(int, float, Vector3[], Vector3[], Vector3[], Vector3[], CollisionLayer, Vector3?)
```

Query water physical data at a set of points. Pass in null to any out parameters that are not required.

## Declaration

```
int Query(int hash, float minimumLength, Vector3[] points, Vector3[] displacements,  
        ↵ Vector3[] normals, Vector3[] velocities, CollisionLayer layer = CollisionLayer.  
        ↵ Everything, Vector3? center = null)
```

## Parameters

hash	Unique ID for calling code. Typically acquired by calling GetHashCode.
minimumLength	The minimum spatial length of the object, such as the width of a boat. Useful for filtering out detail when not needed. Set to zero to get full available detail.
points	The world space points that will be queried.
displacements	Resulting displacement vectors at the query positions. Add sea level to Y to get world space height.
normals	Resulting normals at the query positions. Pass null if this information is not required.
velocities	Resulting velocities at the query positions. Pass null if this information is not required.
layer	The layer this query targets.
center	The center of all the query positions. Used to choose the closest query provider.

## Return

The status of the query.

## 1.52.2 Inherited Methods

### RetrieveSucceeded

`RetrieveSucceeded(int)`

Check if the query results could be retrieved successfully using the return code from Query method.

### Declaration

```
bool RetrieveSucceeded(int status)
```

### Parameters

status	The query status returned from Query.
--------	---------------------------------------

### Return

Whether the retrieve was successful.

## 1.53 IDepthProvider

Interface in `WaveHarmonic.Crest`, `WaveHarmonic.Crest`

Interface for an object that returns water depth and distance to water edge.

### 1.53.1 Methods

#### Query

`Query(int, float, Vector3[], Vector3[], Vector3?)`

Query water depth data at a set of points.

### Declaration

```
int Query(int hash, float minimumLength, Vector3[] points, Vector3[] results, Vector3? center = null)
```

### Parameters

hash	Unique ID for calling code. Typically acquired by calling GetHashCode.
minimumLength	The minimum spatial length of the object, such as the width of a boat. Useful for filtering out detail when not needed. Set to zero to get full available detail.
points	The world space points that will be queried.
results	Water depth and distance to shoreline (XY respectively). Both are signed.
center	The center of all the query positions. Used to choose the closest query provider.

**Return**

The status of the query.

**1.53.2 Inherited Methods****RetrieveSucceeded****RetrieveSucceeded(int)**

Check if the query results could be retrieved successfully using the return code from Query method.

**Declaration**

```
bool RetrieveSucceeded(int status)
```

**Parameters**

status	The query status returned from Query.
--------	---------------------------------------

**Return**

Whether the retrieve was successful.

**1.54 IFlowProvider**

Interface in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Interface for an object that returns water surface displacement and height.

**1.54.1 Methods****Query****Query(int, float, Vector3[], Vector3[], Vector3?)**

Query water flow data (horizontal motion) at a set of points.

**Declaration**

```
int Query(int hash, float minimumLength, Vector3[] points, Vector3[] results, Vector3?  
center = null)
```

## Parameters

hash	Unique ID for calling code. Typically acquired by calling GetHashCode.
minimumLength	The minimum spatial length of the object, such as the width of a boat. Useful for filtering out detail when not needed. Set to zero to get full available detail.
points	The world space points that will be queried.
results	Water surface flow velocities at the query positions.
center	The center of all the query positions. Used to choose the closest query provider.

## Return

The status of the query.

### 1.54.2 Inherited Methods

#### RetrieveSucceeded

##### RetrieveSucceeded(int)

Check if the query results could be retrieved successfully using the return code from Query method.

#### Declaration

```
bool RetrieveSucceeded(int status)
```

## Parameters

status	The query status returned from Query.
--------	---------------------------------------

## Return

Whether the retrieve was successful.

## 1.55 IQueryProvider

Interface in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Base interface for providers.

### 1.55.1 Methods

#### RetrieveSucceeded

##### RetrieveSucceeded(int)

Check if the query results could be retrieved successfully using the return code from Query method.

**Declaration**

```
bool RetrieveSucceeded(int status)
```

**Parameters**

status	The query status returned from Query.
--------	---------------------------------------

**Return**

Whether the retrieve was successful.

## 1.56 ITimeProvider

Interface in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Base class for scripts that provide the time to the water system.

See derived classes for examples.

### 1.56.1 Properties

**Time**

Current time.

**Declaration**

```
float Time { get; }
```

**Delta**

Delta time.

**Declaration**

```
float Delta { get; }
```

## 1.57 LevelGeometryLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *GeometryLodInputData*

Data storage for for the Geometry input mode.

### 1.57.1 Inherited Properties

#### Geometry

Geometry to render into the simulation.

#### Declaration

```
public Mesh Geometry { get; set; }
```

## 1.58 LevelLod

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *Lod*

Data that gives the water level (water height at rest and gradients).

### 1.58.1 Inherited Properties

#### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

#### Declaration

```
public bool Blur { get; set; }
```

#### BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

#### Declaration

```
public int BlurIterations { get; set; }
```

#### Enabled

Whether the simulation is enabled.

#### Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

### Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overridden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will be used.

### Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.59 LevelLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the *LevelLod*.

Attach this to objects that you want to influence the water height.

### 1.59.1 Properties

#### HeightRange

The minimum and maximum height value to report for water chunk culling.

#### Declaration

```
public Vector2 HeightRange { get; set; }
```

#### OverrideHeight

Whether to use the manual “Height Range” for water chunk culling.

Mandatory for non mesh inputs like “Texture”.

#### Declaration

```
public bool OverrideHeight { get; set; }
```

### 1.59.2 Inherited Properties

#### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

#### Declaration

```
public LodInputBlend Blend { get; set; }
```

#### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

#### Declaration

```
public float FeatherWidth { get; set; }
```

#### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

## Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

## Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputMode) to set the mode via scripting. The mode cannot be changed after creation.

## Declaration

```
public LodInputMode Mode { get; }
```

## Queue

The order this input will render.

Order is Queue plus SiblingIndex

## Declaration

```
public int Queue { get; set; }
```

## Weight

Scales the input.

## Declaration

```
public float Weight { get; set; }
```

## ForceRenderingOff

Disables rendering of input into data, but continues most scripting activities.

## Declaration

```
public bool ForceRenderingOff { get; set; }
```

## Data

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.59.3 Inherited Methods****GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

**1.60 LevelRendererLodInputData**

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

**1.60.1 Inherited Properties****CheckShaderName**

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

**Declaration**

```
public bool CheckShaderName { get; set; }
```

**CheckShaderPasses**

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

**Declaration**

```
public bool CheckShaderPasses { get; set; }
```

**DisableRenderer**

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

**Declaration**

```
public bool DisableRenderer { get; set; }
```

**OverrideShaderPass**

Whether to set the shader pass manually.

**Declaration**

```
public bool OverrideShaderPass { get; set; }
```

**Renderer**

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

**Declaration**

```
public Renderer Renderer { get; set; }
```

**ShaderPassIndex**

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.61 LevelTextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *TextureLodInputData*

Data storage for for the Texture input mode.

### 1.61.1 Inherited Properties

**Multiplier**

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

**Texture**

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.62 Lod

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Base class for data/behaviours created on each LOD.

### 1.62.1 Properties

**Blur**

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

**Declaration**

```
public bool Blur { get; set; }
```

**BlurIterations**

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

**Declaration**

```
public int BlurIterations { get; set; }
```

**Enabled**

Whether the simulation is enabled.

**Declaration**

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

### Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

### Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.63 Lod<T>

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [Lod](#)

Base type for simulations with a provider.

### 1.63.1 Properties

#### MaximumQueryCount

Maximum number of queries that can be performed when using GPU queries.

#### Declaration

```
public int MaximumQueryCount { get; }
```

#### QuerySource

Where to obtain water data on CPU for physics / gameplay.

#### Declaration

```
public LodQuerySource QuerySource { get; }
```

#### Provider

Provides data from the GPU to CPU.

#### Declaration

```
public T Provider { get; set; }
```

### 1.63.2 Inherited Properties

#### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

#### Declaration

```
public bool Blur { get; set; }
```

#### BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

#### Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

### Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

### Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

## Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

# 1.64 LodInput

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Base class for scripts that register inputs to the various LOD data types.

## 1.64.1 Properties

### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

### Declaration

```
public LodInputBlend Blend { get; set; }
```

### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

### Declaration

```
public float FeatherWidth { get; set; }
```

### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

**Declaration**

```
public LodInputMode Mode { get; }
```

**Queue**

The order this input will render.

Order is Queue plus SiblingIndex

**Declaration**

```
public int Queue { get; set; }
```

**Weight**

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.64.2 Methods****GetData**

### GetData<T>()

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

#### Declaration

```
public T GetData<T>() where T : LodInputData
```

#### Return

The casted data.

## 1.65 LodInputBlend

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Blend presets for inputs.

### 1.65.1 Properties

#### Off

No blending. Overwrites.

#### Declaration

```
Off = 0
```

#### Additive

Additive blending.

#### Declaration

```
Additive = 1
```

#### Minimum

Takes the minimum value.

#### Declaration

```
Minimum = 2
```

#### Maximum

Takes the maximum value.

**Declaration**

```
Maximum = 3
```

**Alpha**

Applies the inverse weight to the target.

Basically overwrites what is already in the simulation.

**Declaration**

```
Alpha = 4
```

**AlphaClip**

Same as alpha except anything above zero will overwrite rather than blend.

**Declaration**

```
AlphaClip = 5
```

## 1.66 LodInputData

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Data storage for an input, pertinent to the associated input mode.

## 1.67 LodInputModule

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Modes that inputs can use. Not all inputs support all modes. Refer to the UI.

### 1.67.1 Properties

**Unset**

Unset is the serialization default.

This will be replaced with the default mode automatically. Unset can also be used if something is invalid.

**Declaration**

```
Unset = 0
```

**Paint**

Hand-painted data by the user.

**Declaration**

```
Paint = 1
```

**Spline**

Driven by a user created spline.

**Declaration**

```
Spline = 2
```

**Renderer**

Attached ‘Renderer’ (mesh, particle or other) used to drive data.

**Declaration**

```
Renderer = 3
```

**Primitive**

Driven by a mathematical primitive such as a cube or sphere.

**Declaration**

```
Primitive = 4
```

**Global**

Covers the entire water area.

**Declaration**

```
Global = 5
```

**Texture**

Data driven by a user provided texture.

**Declaration**

```
Texture = 6
```

## Geometry

Renders geometry using a default material.

### Declaration

```
Geometry = 7
```

## 1.68 LodInputPrimitive

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Primitive shapes.

### 1.68.1 Properties

#### Sphere

Spheroid.

#### Declaration

```
Sphere = 0
```

#### Cube

Cuboid.

#### Declaration

```
Cube = 3
```

#### Quad

Quad.

#### Declaration

```
Quad = 5
```

## 1.69 LodQuerySource

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

The source of collisions (ie water shape).

## 1.69.1 Properties

### None

No query source.

### Declaration

```
None = 0
```

### GPU

Uses AsyncGPUReadback to retrieve data from GPU to CPU.

This is the most optimal approach.

### Declaration

```
GPU = 1
```

### CPU

Computes data entirely on the CPU.

### Declaration

```
CPU = 2
```

## 1.70 LodSettings

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Base class for simulation settings.

## 1.71 LodTextureFormatMode

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Texture format preset.

## 1.71.1 Properties

### Manual

Uses the Texture Format property.

## Declaration

```
Manual = 0
```

## Performance

Chooses a texture format for performance.

## Declaration

```
Performance = 100
```

## Precision

Chooses a texture format for precision.

This format can reduce artifacts.

## Declaration

```
Precision = 200
```

## Automatic

Chooses a texture format based on another.

For example, Dynamic Waves will match precision of Animated Waves.

## Declaration

```
Automatic = 300
```

# 1.72 Meniscus

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Renders the meniscus (waterline).

## 1.72.1 Properties

### CameraExclusions

Rules to exclude cameras from rendering the meniscus.

These are exclusion rules, so for all cameras, select Nothing. These rules are applied on top of the Layer rules.

## Declaration

```
public WaterCameraExclusion CameraExclusions { get; set; }
```

### Enabled

Whether the meniscus is enabled.

#### Declaration

```
public bool Enabled { get; set; }
```

### Layer

Any camera with this layer in its culling mask will render the meniscus.

#### Declaration

```
public int Layer { get; set; }
```

### Material

The meniscus material.

#### Declaration

```
public Material Material { get; set; }
```

### ForceRenderingOff

Disables rendering without de-allocating.

#### Declaration

```
public bool ForceRenderingOff { get; set; }
```

## 1.73 NetworkedTimeProvider

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [TimeProvider](#), [ITimeProvider](#)

Gives a time to Crest with a custom time offset.

Assign this component to the [WaterRenderer](#) component and set the NetworkedTimeProvider.TimeOffsetToServer property of this component to the delta from this client's time to the shared server time.

### 1.73.1 Properties

#### TimeOffsetToServer

If Time.time on this client is 1.5s ahead of the shared/server Time.time, set this field to -1.5.

**Declaration**

```
public float TimeOffsetToServer { get; set; }
```

**Time**

Current time.

**Declaration**

```
public override float Time { get; }
```

**Delta**

Delta time.

**Declaration**

```
public override float Delta { get; }
```

## 1.74 PersistentLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [Lod](#)

A persistent simulation that moves around with a displacement LOD.

### 1.74.1 Properties

**SimulationFrequency**

Frequency to run the simulation, in updates per second.

Lower frequencies are more efficient but may lead to visible jitter or slowness.

**Declaration**

```
public int SimulationFrequency { get; set; }
```

### 1.74.2 Inherited Properties

**Blur**

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

## Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

## Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

## Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

## Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

## Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

## Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

### Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

# 1.75 Placement

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest

How a component is placed in the world.

## 1.75.1 Properties

### Fixed

The component is in a fixed position.

### Declaration

```
Fixed = 0
```

### Transform

The component follows the transform.

### Declaration

```
Transform = 1
```

### Viewpoint

The component follows the viewpoint.

### Declaration

```
Viewpoint = 2
```

## 1.76 PortalMode

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest.Portals

Portal rendering modes.

### 1.76.1 Properties

#### Portal

A portal to infinite water, rendered from front faces of geometry.

#### Declaration

```
Portal = 2
```

#### Volume

A volume of water rendered which only works when the viewer is outside the volume.

It uses both front faces and back faces. It is more efficient than VolumeFlyThrough.

#### Declaration

```
Volume = 3
```

#### VolumeFlyThrough

A volume of water rendered which also works with the viewer inside the volume.

It uses both front faces and back faces. It also requires the stencil buffer and is less efficient than Volume.

#### Declaration

```
VolumeFlyThrough = 4
```

#### Tunnel

Removes the water surface and underwater effect for caves etc.

The walls must be covered by geometry (eg cave) for this to look correct.

#### Declaration

```
Tunnel = 5
```

## 1.77 PortalRenderer

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest.Portals

This renderer can remove water inside or outside of geometry.

## 1.77.1 Properties

### Active

Whether the *PortalRenderer* is active (enabled and valid).

#### Declaration

```
public bool Active { get; }
```

### Enabled

Whether portal rendering is enabled.

#### Declaration

```
public bool Enabled { get; set; }
```

### Geometry

Mesh (Mesh Filter) to use to render the portal.

It will use the Mesh Filter's transform.

#### Declaration

```
public MeshFilter Geometry { get; set; }
```

### Invert

Use the back-faces of the mesh.

Useful for portholes on watercraft or tunnelling through water.

#### Declaration

```
public bool Invert { get; set; }
```

### Mode

Rendering mode of the portal (and water surface).

See the manual for more details.

#### Declaration

```
public PortalMode Mode { get; set; }
```

## 1.78 QualitySettingsOverride

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Overrides global quality settings.

## 1.78.1 Properties

### LodBias

Overrides the LOD bias for meshes.

Highest quality is infinity.

### Declaration

```
public float LodBias { get; set; }
```

### MaximumLodLevel

Overrides the maximum LOD level.

Highest quality is zero.

### Declaration

```
public int MaximumLodLevel { get; set; }
```

### OverrideLodBias

Whether to override the LOD bias.

### Declaration

```
public bool OverrideLodBias { get; set; }
```

### OverrideMaximumLodLevel

Whether to override the maximum LOD level.

### Declaration

```
public bool OverrideMaximumLodLevel { get; set; }
```

### OverrideTerrainPixelError

Whether to override the terrain pixel error.

### Declaration

```
public bool OverrideTerrainPixelError { get; set; }
```

### TerrainPixelError

Overrides the pixel error value for terrains.

Highest quality is zero.

**Declaration**

```
public float TerrainPixelError { get; set; }
```

## 1.79 QueryEvents

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Emits events (UnityEvents) based on the sampled water data.

### 1.79.1 Properties

**DistanceFromEdge**

Sends the distance from the water's edge.

**Declaration**

```
public Action<float> DistanceFromEdge { get; set; }
```

**DistanceFromEdgeCurve**

Apply a curve to the distance.

Values towards “one” means closer to the water’s edge.

**Declaration**

```
public AnimationCurve DistanceFromEdgeCurve { get; set; }
```

**DistanceFromEdgeMaximum**

The maximum distance.

Always use a real distance in real units (ie not normalized).

**Declaration**

```
public float DistanceFromEdgeMaximum { get; set; }
```

**DistanceFromEdgeSigned**

Whether to keep the sign of the value (ie positive/negative).

A positive value means the query point is over water, while a negative means it is over land.

**Declaration**

```
public bool DistanceFromEdgeSigned { get; set; }
```

### DistanceFromEdgeUseCurve

Apply a curve to the distance.

Normalizes and inverts the distance to be between zero and one, then applies a curve.

#### Declaration

```
public bool DistanceFromEdgeUseCurve { get; set; }
```

### DistanceFromSurface

Sends the distance from the water surface.

#### Declaration

```
public Action<float> DistanceFromSurface { get; set; }
```

### DistanceFromSurfaceCurve

Apply a curve to the distance.

Values towards “one” means closer to the water surface.

#### Declaration

```
public AnimationCurve DistanceFromSurfaceCurve { get; set; }
```

### DistanceFromSurfaceMaximum

The maximum distance.

Always use a real distance in real units (ie not normalized).

#### Declaration

```
public float DistanceFromSurfaceMaximum { get; set; }
```

### DistanceFromSurfaceSigned

Whether to keep the sign of the value (ie positive/negative).

A positive value means the query point is above the surface, while a negative means it below the surface.

#### Declaration

```
public bool DistanceFromSurfaceSigned { get; set; }
```

### DistanceFromSurfaceUseCurve

Whether to apply a curve to the distance.

Normalizes and inverts the distance to be between zero and one, then applies a curve.

## Declaration

```
public bool DistanceFromSurfaceUseCurve { get; set; }
```

## Layer

Which water collision layer to target.

## Declaration

```
public CollisionLayer Layer { get; set; }
```

## MinimumWavelength

The minimum wavelength for queries.

The higher the value, the more smaller waves will be ignored when sampling the water surface.

## Declaration

```
public float MinimumWavelength { get; set; }
```

## Source

What transform should the queries be based on.

“Viewer” will reuse queries already performed by the Water Renderer

## Declaration

```
public QuerySource Source { get; set; }
```

## Viewer

The viewer as the source of the queries.

Only needs to be set if using multiple viewpoints on the Water Renderer.

## Declaration

```
public Camera Viewer { get; set; }
```

## 1.79.2 Events

### OnAboveWater

Triggers when game object goes above water surface.

Triggers once per state change.

**Declaration**

```
public Action OnAboveWater { get; set; }
```

**OnBelowWater**

Triggers when game object goes below water surface.

Triggers once per state change.

**Declaration**

```
public Action OnBelowWater { get; set; }
```

## 1.80 QuerySource

Enum in *WaveHarmonic.Crest*, WaveHarmonic.Crest

What transform to use for queries.

### 1.80.1 Properties

**Transform**

This game object's transform.

**Declaration**

```
Transform = 0
```

**Viewer**

The viewer's transform.

The viewer is the main camera the system uses.

**Declaration**

```
Viewer = 1
```

## 1.81 RayCastHelper

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Helper to trace a ray against the water surface.

Works by sampling at a set of points along the ray and interpolating the intersection location.

### 1.81.1 Constructors

#### RayCastHelper

`RayCastHelper(float, float)`

The length of the ray and the step size must be given here. The smaller the step size, the greater the accuracy.

#### Declaration

```
public RayCastHelper(float rayLength, float rayStepSize = 2)
```

#### Parameters

rayLength	Length of the ray.
rayStepSize	Size of the step. With length the number of steps is computed.

### 1.81.2 Methods

#### RayCast

`RayCast(Vector3, Vector3, out float, CollisionLayer)`

Call this once each frame to do the query.

#### Declaration

```
public bool RayCast(Vector3 origin, Vector3 direction, out float distance,  
CollisionLayer layer = CollisionLayer.Everything)
```

#### Parameters

origin	World space position of ray origin
direction	World space ray direction
distance	The distance along the ray to the first intersection with the water surface.
layer	The layer this ray targets.

#### Return

True if the results have come back from the GPU, and if the ray intersects the water surface.

## 1.82 RendererLodInputData

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInputData](#)

Data storage for the Renderer input mode.

## 1.82.1 Properties

### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

#### Declaration

```
public bool CheckShaderName { get; set; }
```

### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

#### Declaration

```
public bool CheckShaderPasses { get; set; }
```

### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

#### Declaration

```
public bool DisableRenderer { get; set; }
```

### OverrideShaderPass

Whether to set the shader pass manually.

#### Declaration

```
public bool OverrideShaderPass { get; set; }
```

### Renderer

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

#### Declaration

```
public Renderer Renderer { get; set; }
```

### ShaderPassIndex

The shader pass to execute.

Set to -1 to execute all passes.

## Declaration

```
public int ShaderPassIndex { get; set; }
```

## 1.83 SampleCollisionHelper

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Helper to obtain the water surface collision at a single point per frame.

It is not particularly efficient to sample a single point, but is a fairly common case.

### 1.83.1 Methods

#### Sample

```
Sample(Vector3, out Vector3, out Vector3, out Vector3, float, CollisionLayer)
```

 **Obsolete**

Please use SampleDisplacement instead. Be wary that the new API has switch the normal parameter with velocity.

Sample displacement data.

Only call once per frame.

#### Declaration

```
public bool Sample(Vector3 position, out Vector3 displacement, out Vector3 normal, out_
    ↪Vector3 velocity, float minimumLength = 0, CollisionLayer layer = CollisionLayer.
    ↪Everything)
```

#### Parameters

position	World space position to sample.
displacement	The water surface displacement to point.
normal	The water surface normal.
velocity	The velocity of the water surface excluding flow velocity.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

#### Return

Whether the query was successful.

```
Sample(Vector3, out float, out Vector3, out Vector3, float, CollisionLayer)
```

 **Obsolete**

Please use SampleHeight instead. Be wary that the new API has switch the normal parameter with velocity.

Sample displacement data.

Only call once per frame.

### Declaration

```
public bool Sample(Vector3 position, out float height, out Vector3 normal, out Vector3
    ↵velocity, float minimumLength = 0, CollisionLayer layer = CollisionLayer.Everything)
```

### Parameters

position	World space position to sample.
height	The water surface height.
normal	The water surface normal.
velocity	The velocity of the water surface excluding flow velocity.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

### Return

Whether the query was successful.

```
Sample(Vector3, out float, out Vector3, float, CollisionLayer)
```

 **Obsolete**

Please use SampleHeight instead. Be wary that the new API has switch the normal parameter with velocity.

Sample displacement data.

Only call once per frame.

### Declaration

```
public bool Sample(Vector3 position, out float height, out Vector3 normal, float
    ↵minimumLength = 0, CollisionLayer layer = CollisionLayer.Everything)
```

## Parameters

position	World space position to sample.
height	The water surface height.
normal	The water surface normal.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

## Return

Whether the query was successful.

`Sample(Vector3, out float, float, CollisionLayer)`

### Obsolete

Please use SampleHeight instead. Be wary that the new API has switch the normal parameter with velocity.

Sample displacement data.

Only call once per frame.

## Declaration

```
public bool Sample(Vector3 position, out float height, float minimumLength = 0, 
CollisionLayer layer = CollisionLayer.Everything)
```

## Parameters

position	World space position to sample.
height	The water surface height.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

## Return

Whether the query was successful.

**SampleDisplacement**

`SampleDisplacement(Vector3, out Vector3, out Vector3, out Vector3, float, CollisionLayer)`

Sample displacement data.

Only call once per frame.

## Declaration

```
public bool SampleDisplacement(Vector3 position, out Vector3 displacement, out Vector3_
    ↪velocity, out Vector3 normal, float minimumLength = 0, CollisionLayer layer =_
    ↪CollisionLayer.Everything)
```

## Parameters

position	World space position to sample.
displacement	The water surface displacement to point.
velocity	The velocity of the water surface excluding flow velocity.
normal	The water surface normal.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

## Return

Whether the query was successful.

`SampleDisplacement(Vector3, out Vector3, out Vector3, float, CollisionLayer)`

Sample displacement data.

Only call once per frame.

## Declaration

```
public bool SampleDisplacement(Vector3 position, out Vector3 displacement, out Vector3_
    ↪velocity, float minimumLength = 0, CollisionLayer layer = CollisionLayer.Everything)
```

## Parameters

position	World space position to sample.
displacement	The water surface displacement to point.
velocity	The velocity of the water surface excluding flow velocity.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

## Return

Whether the query was successful.

**SampleDisplacement(Vector3, out Vector3, float, CollisionLayer)**

Sample displacement data.

Only call once per frame.

**Declaration**

```
public bool SampleDisplacement(Vector3 position, out Vector3 displacement, float  
↳minimumLength = 0, CollisionLayer layer = CollisionLayer.Everything)
```

**Parameters**

position	World space position to sample.
displacement	The water surface displacement to point.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

**Return**

Whether the query was successful.

**SampleHeight****SampleHeight(Vector3, out float, out Vector3, out Vector3, float, CollisionLayer)**

Sample displacement data.

Only call once per frame.

**Declaration**

```
public bool SampleHeight(Vector3 position, out float height, out Vector3 velocity, out  
↳Vector3 normal, float minimumLength = 0, CollisionLayer layer = CollisionLayer.  
↳Everything)
```

**Parameters**

position	World space position to sample.
height	The water surface height.
velocity	The velocity of the water surface excluding flow velocity.
normal	The water surface normal.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

## Return

Whether the query was successful.

`SampleHeight(Vector3, out float, out Vector3, float, CollisionLayer)`

Sample displacement data.

Only call once per frame.

## Declaration

```
public bool SampleHeight(Vector3 position, out float height, out Vector3 velocity, float
    ↵minimumLength = 0, CollisionLayer layer = CollisionLayer.Everything)
```

## Parameters

position	World space position to sample.
height	The water surface height.
velocity	The velocity of the water surface excluding flow velocity.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

## Return

Whether the query was successful.

`SampleHeight(Vector3, out float, float, CollisionLayer)`

Sample displacement data.

Only call once per frame.

## Declaration

```
public bool SampleHeight(Vector3 position, out float height, float minimumLength = 0,
    ↵CollisionLayer layer = CollisionLayer.Everything)
```

## Parameters

position	World space position to sample.
height	The water surface height.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.
layer	The collision layer to target.

**Return**

Whether the query was successful.

## 1.84 SampleDepthHelper

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Helper to obtain the depth data at a single point.

### 1.84.1 Methods

#### SampleDistanceToWaterEdge

`SampleDistanceToWaterEdge(Vector3, out float)`

Sample water edge distance.

Only call once per frame.

#### Declaration

```
public bool SampleDistanceToWaterEdge(Vector3 position, out float distance)
```

#### Parameters

position	World space position to sample.
distance	Filled by the distance to water edge at the query position.

**Return**

Whether the query was successful.

## 1.85 SampleFlowHelper

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Helper to obtain the flow data (horizontal water motion) at a single point.

It is not particularly efficient to sample a single point, but is a fairly common case.

### 1.85.1 Methods

#### Sample

`Sample(Vector3, out Vector2, float)`

Sample flow data.

Only call once per frame.

## Declaration

```
public bool Sample(Vector3 position, out Vector2 flow, float minimumLength = 0)
```

## Parameters

position	World space position to sample.
flow	Filled with resulting flow.
minimumLength	The smallest length scale you are interested in. If you are sampling data for boat physics, pass in the boats width. Larger objects will ignore smaller details in the data.

## Return

Whether the query was successful.

## 1.86 ScatteringLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [ColorLod](#)

Overrides the main scattering color.

### 1.86.1 Inherited Properties

#### ShorelineColor

Color of the shoreline color.

#### Declaration

```
public Color ShorelineColor { get; set; }
```

#### ShorelineColorFalloff

Shoreline color falloff value.

#### Declaration

```
public float ShorelineColorFalloff { get; set; }
```

#### ShorelineColorMaximumDistance

The maximum distance of the shoreline color.

If using Depth, then it is maximum depth.

## Declaration

```
public float ShorelineColorMaximumDistance { get; set; }
```

## ShorelineColorSource

Source of the shoreline color.

## Declaration

```
public ShorelineVolumeColorSource ShorelineColorSource { get; set; }
```

## Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

## Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

## Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

## Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

## Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

### Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overriden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

### Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

### Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

## 1.87 ScatteringLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [ScatteringLod](#).

Attach this to objects that you want to influence the scattering color.

### 1.87.1 Inherited Properties

#### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

## Declaration

```
public LodInputBlend Blend { get; set; }
```

## FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

## Declaration

```
public float FeatherWidth { get; set; }
```

## FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

## Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

## Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

## Declaration

```
public LodInputModule Mode { get; }
```

## Queue

The order this input will render.

Order is Queue plus SiblingIndex

## Declaration

```
public int Queue { get; set; }
```

## Weight

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *Shape-WavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.87.2 Inherited Methods****GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

**1.88 ScatteringRendererLodInputData**

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

### 1.88.1 Inherited Properties

#### CheckShaderName

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

#### Declaration

```
public bool CheckShaderName { get; set; }
```

#### CheckShaderPasses

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

#### Declaration

```
public bool CheckShaderPasses { get; set; }
```

#### DisableRenderer

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

#### Declaration

```
public bool DisableRenderer { get; set; }
```

#### OverrideShaderPass

Whether to set the shader pass manually.

#### Declaration

```
public bool OverrideShaderPass { get; set; }
```

#### Renderer

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

#### Declaration

```
public Renderer Renderer { get; set; }
```

#### ShaderPassIndex

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.89 ScatteringTextureLodInputData

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [TextureLodInputData](#)

Data storage for for the Texture input mode.

### 1.89.1 Inherited Properties

**Multiplier**

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

**Texture**

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.90 ShadowLod

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [PersistentLod](#)

Stores shadowing data to use during water shading.

Shadowing is persistent and supports sampling across many frames and jittered sampling for (very) soft shadows. Soft shadows is red, hard shadows is green.

### 1.90.1 Properties

**CurrentFrameWeightHard**

Current frame weight for accumulation over frames for hard shadows.

Roughly means ‘responsiveness’ for hard shadows.

## Declaration

```
public float CurrentFrameWeightHard { get; set; }
```

## CurrentFrameWeightSoft

Current frame weight for accumulation over frames for soft shadows.

Roughly means ‘responsiveness’ for soft shadows.

## Declaration

```
public float CurrentFrameWeightSoft { get; set; }
```

## JitterDiameterHard

Jitter diameter for hard shadows, controls softness of this shadowing component.

## Declaration

```
public float JitterDiameterHard { get; set; }
```

## JitterDiameterSoft

Jitter diameter for soft shadows, controls softness of this shadowing component.

## Declaration

```
public float JitterDiameterSoft { get; set; }
```

## 1.90.2 Inherited Properties

### SimulationFrequency

Frequency to run the simulation, in updates per second.

Lower frequencies are more efficient but may lead to visible jitter or slowness.

## Declaration

```
public int SimulationFrequency { get; set; }
```

### Blur

Blurs the output.

Enable if blurring is desired or intolerable artifacts are present. The blur is optimized to only run on inner LODs and at lower scales.

## Declaration

```
public bool Blur { get; set; }
```

## BlurIterations

Number of blur iterations.

Blur iterations are optimized to only run maximum iterations on the inner LODs.

### Declaration

```
public int BlurIterations { get; set; }
```

## Enabled

Whether the simulation is enabled.

### Declaration

```
public bool Enabled { get; set; }
```

## OverrideResolution

Whether to override the resolution.

If not enabled, then the simulation will use the resolution defined on the Water Renderer.

### Declaration

```
public bool OverrideResolution { get; set; }
```

## Resolution

The resolution of the simulation data.

Set higher for sharper results at the cost of higher memory usage.

### Declaration

```
public int Resolution { get; set; }
```

## TextureFormat

The render texture format used for this simulation data.

It will be overridden if the format is incompatible with the platform.

### Declaration

```
public GraphicsFormat TextureFormat { get; set; }
```

## TextureFormatMode

Chooses a texture format based on a preset value.

## Declaration

```
public LodTextureFormatMode TextureFormatMode { get; set; }
```

## CompatibleTextureFormat

This is the platform compatible texture format that will used.

## Declaration

```
public GraphicsFormat CompatibleTextureFormat { get; }
```

# 1.91 ShadowLodInput

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [LodInput](#)

Registers a custom input to the [ShadowLod](#).

Attach this objects that you want use to override shadows.

## 1.91.1 Inherited Properties

### Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

## Declaration

```
public LodInputBlend Blend { get; set; }
```

### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

## Declaration

```
public float FeatherWidth { get; set; }
```

### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

## Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

## Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputMode) to set the mode via scripting. The mode cannot be changed after creation.

## Declaration

```
public LodInputMode Mode { get; }
```

## Queue

The order this input will render.

Order is Queue plus SiblingIndex

## Declaration

```
public int Queue { get; set; }
```

## Weight

Scales the input.

## Declaration

```
public float Weight { get; set; }
```

## ForceRenderingOff

Disables rendering of input into data, but continues most scripting activities.

## Declaration

```
public bool ForceRenderingOff { get; set; }
```

## Data

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.91.2 Inherited Methods****GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

**1.92 ShadowRendererLodInputData**

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

**1.92.1 Inherited Properties****CheckShaderName**

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

**Declaration**

```
public bool CheckShaderName { get; set; }
```

**CheckShaderPasses**

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

**Declaration**

```
public bool CheckShaderPasses { get; set; }
```

**DisableRenderer**

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

**Declaration**

```
public bool DisableRenderer { get; set; }
```

**OverrideShaderPass**

Whether to set the shader pass manually.

**Declaration**

```
public bool OverrideShaderPass { get; set; }
```

**Renderer**

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

**Declaration**

```
public Renderer Renderer { get; set; }
```

**ShaderPassIndex**

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.93 ShapeFFT

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *ShapeWaves*

FFT wave shape.

### 1.93.1 Properties

**MaximumHorizontalDisplacement**

Maximum amount a point on the surface will be displaced horizontally by waves from its rest position.

Increase this if gaps appear at sides of screen.

## Declaration

```
public float MaximumHorizontalDisplacement { get; set; }
```

## MaximumVerticalDisplacement

Maximum amount the surface will be displaced vertically from sea level.

Increase this if gaps appear at bottom of screen.

## Declaration

```
public float MaximumVerticalDisplacement { get; set; }
```

## OverrideCulling

Whether to override automatic culling based on heuristics.

## Declaration

```
public bool OverrideCulling { get; set; }
```

## OverrideGlobalWindTurbulence

Whether to use the wind turbulence on this component rather than the global wind turbulence.

Global wind turbulence comes from the Water Renderer component.

## Declaration

```
public bool OverrideGlobalWindTurbulence { get; set; }
```

## TimeLoopLength

FFT waves will loop with a period of this many seconds.

## Declaration

```
public float TimeLoopLength { get; set; }
```

## WindAlignment

How aligned the waves are with wind.

## Declaration

```
public float WindAlignment { get; set; }
```

## WindTurbulence

How turbulent/chaotic the waves are.

### Declaration

```
public float WindTurbulence { get; set; }
```

## 1.93.2 Inherited Properties

### EvaluateSpectrumAtRunTimeEveryFrame

Whether to evaluate the spectrum every frame.

When false, the wave spectrum is evaluated once on startup in editor play mode and standalone builds, rather than every frame. This is less flexible, but it reduces the performance cost significantly.

### Declaration

```
public bool EvaluateSpectrumAtRunTimeEveryFrame { get; set; }
```

### IncludeInDropDetailHeightBasedOnWaves

Whether the maximum possible vertical displacement is used for the Drop Detail Height Based On Waves calculation.

This setting is ignored for global waves, as they always contribute. For local waves, only enable for large areas that are treated like global waves (eg a storm).

### Declaration

```
public bool IncludeInDropDetailHeightBasedOnWaves { get; set; }
```

### OverrideGlobalWindDirection

Whether to use the wind direction on this component rather than the global wind direction.

Global wind direction comes from the Water Renderer component.

### Declaration

```
public bool OverrideGlobalWindDirection { get; set; }
```

### OverrideGlobalWindSpeed

Whether to use the wind speed on this component rather than the global wind speed.

Global wind speed comes from the Water Renderer component.

### Declaration

```
public bool OverrideGlobalWindSpeed { get; set; }
```

## Resolution

Resolution to use for wave generation buffers.

Low resolutions are more efficient but can result in noticeable patterns in the shape.

## Declaration

```
public int Resolution { get; set; }
```

## RespectShallowWaterAttenuation

How much these waves respect the shallow water attenuation.

Attenuation is defined on the Animated Waves. Set to zero to ignore attenuation.

## Declaration

```
public float RespectShallowWaterAttenuation { get; set; }
```

## SeaLevelOnly

Whether global waves is applied above or below sea level.

Waves are faded out to avoid hard transitions. They are fully faded by 1m from sea level.

## Declaration

```
public bool SeaLevelOnly { get; set; }
```

## Spectrum

The spectrum that defines the water surface shape.

## Declaration

```
public WaveSpectrum Spectrum { get; set; }
```

## WaveDirectionHeadingAngle

Primary wave direction heading (degrees).

This is the angle from x axis in degrees that the waves are oriented towards. If a spline is being used to place the waves, this angle is relative to the spline.

## Declaration

```
public float WaveDirectionHeadingAngle { get; set; }
```

## WindSpeed

Wind speed in km/h. Controls wave conditions.

## Declaration

```
public float WindSpeed { get; set; }
```

## WindSpeedKPH

The wind speed in kilometers per hour (KPH).

Wind speed can come from this component or the *WaterRenderer*.

## Declaration

```
public float WindSpeedKPH { get; }
```

## WindSpeedMPS

The wind speed in meters per second (MPS).

Wind speed can come from this component or the *WaterRenderer*.

## Declaration

```
public float WindSpeedMPS { get; }
```

## Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

## Declaration

```
public LodInputBlend Blend { get; set; }
```

## FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

## Declaration

```
public float FeatherWidth { get; set; }
```

## FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

## Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

## Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputMode) to set the mode via scripting. The mode cannot be changed after creation.

## Declaration

```
public LodInputMode Mode { get; }
```

## Queue

The order this input will render.

Order is Queue plus SiblingIndex

## Declaration

```
public int Queue { get; set; }
```

## Weight

Scales the input.

## Declaration

```
public float Weight { get; set; }
```

## ForceRenderingOff

Disables rendering of input into data, but continues most scripting activities.

## Declaration

```
public bool ForceRenderingOff { get; set; }
```

## Data

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

### 1.93.3 Inherited Methods

**GetData****GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

## 1.94 ShapeGerstner

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *ShapeWaves*

Gerstner wave shape.

### 1.94.1 Properties

**\_Wavelengths**

Wavelengths. Requires Manual Generation to be enabled.

**Declaration**

```
public float[] _Wavelengths
```

**\_Amplitudes**

Amplitudes. Requires Manual Generation to be enabled.

**Declaration**

```
public float[] _Amplitudes
```

## \_Powers

Powers. Requires Manual Generation to be enabled.

### Declaration

```
public float[] _Powers
```

## \_AngleDegrees

Angles. Requires Manual Generation to be enabled.

### Declaration

```
public float[] _AngleDegrees
```

## \_Phases

Phases. Requires Manual Generation to be enabled.

### Declaration

```
public float[] _Phases
```

## ComponentsPerOctave

How many wave components to generate in each octave.

### Declaration

```
public int ComponentsPerOctave { get; set; }
```

## ManualGeneration

Prevent data arrays from being written to so one can provide their own.

### Declaration

```
public bool ManualGeneration { get; set; }
```

## RandomSeed

Change to get a different set of waves.

### Declaration

```
public int RandomSeed { get; set; }
```

## ReverseWaveWeight

The weight of the opposing, second pair of Gerstner waves.

Each Gerstner wave is actually a pair of waves travelling in opposite directions (similar to FFT). This weight is applied to the wave travelling in against-wind direction. Set to zero to obtain simple single waves which are useful for shorelines waves.

### Declaration

```
public float ReverseWaveWeight { get; set; }
```

## Swell

Use a swell spectrum as the default.

Uses a swell spectrum as default (when none is assigned), and disabled reverse waves.

### Declaration

```
public bool Swell { get; set; }
```

## 1.94.2 Inherited Properties

### EvaluateSpectrumAtRunTimeEveryFrame

Whether to evaluate the spectrum every frame.

When false, the wave spectrum is evaluated once on startup in editor play mode and standalone builds, rather than every frame. This is less flexible, but it reduces the performance cost significantly.

### Declaration

```
public bool EvaluateSpectrumAtRunTimeEveryFrame { get; set; }
```

### IncludeInDropDetailHeightBasedOnWaves

Whether the maximum possible vertical displacement is used for the Drop Detail Height Based On Waves calculation.

This setting is ignored for global waves, as they always contribute. For local waves, only enable for large areas that are treated like global waves (eg a storm).

### Declaration

```
public bool IncludeInDropDetailHeightBasedOnWaves { get; set; }
```

### OverrideGlobalWindDirection

Whether to use the wind direction on this component rather than the global wind direction.

Global wind direction comes from the Water Renderer component.

## Declaration

```
public bool OverrideGlobalWindDirection { get; set; }
```

## OverrideGlobalWindSpeed

Whether to use the wind speed on this component rather than the global wind speed.

Global wind speed comes from the Water Renderer component.

## Declaration

```
public bool OverrideGlobalWindSpeed { get; set; }
```

## Resolution

Resolution to use for wave generation buffers.

Low resolutions are more efficient but can result in noticeable patterns in the shape.

## Declaration

```
public int Resolution { get; set; }
```

## RespectShallowWaterAttenuation

How much these waves respect the shallow water attenuation.

Attenuation is defined on the Animated Waves. Set to zero to ignore attenuation.

## Declaration

```
public float RespectShallowWaterAttenuation { get; set; }
```

## SeaLevelOnly

Whether global waves is applied above or below sea level.

Waves are faded out to avoid hard transitions. They are fully faded by 1m from sea level.

## Declaration

```
public bool SeaLevelOnly { get; set; }
```

## Spectrum

The spectrum that defines the water surface shape.

## Declaration

```
public WaveSpectrum Spectrum { get; set; }
```

## WaveDirectionHeadingAngle

Primary wave direction heading (degrees).

This is the angle from x axis in degrees that the waves are oriented towards. If a spline is being used to place the waves, this angle is relative to the spline.

### Declaration

```
public float WaveDirectionHeadingAngle { get; set; }
```

## WindSpeed

Wind speed in km/h. Controls wave conditions.

### Declaration

```
public float WindSpeed { get; set; }
```

## WindSpeedKPH

The wind speed in kilometers per hour (KPH).

Wind speed can come from this component or the [WaterRenderer](#).

### Declaration

```
public float WindSpeedKPH { get; }
```

## WindSpeedMPS

The wind speed in meters per second (MPS).

Wind speed can come from this component or the [WaterRenderer](#).

### Declaration

```
public float WindSpeedMPS { get; }
```

## Blend

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

### Declaration

```
public LodInputBlend Blend { get; set; }
```

### FeatherWidth

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

#### Declaration

```
public float FeatherWidth { get; set; }
```

### FollowHorizontalWaveMotion

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

#### Declaration

```
public bool FollowHorizontalWaveMotion { get; set; }
```

### Mode

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

#### Declaration

```
public LodInputModule Mode { get; }
```

### Queue

The order this input will render.

Order is Queue plus SiblingIndex

#### Declaration

```
public int Queue { get; set; }
```

### Weight

Scales the input.

#### Declaration

```
public float Weight { get; set; }
```

## ForceRenderingOff

Disables rendering of input into data, but continues most scripting activities.

### Declaration

```
public bool ForceRenderingOff { get; set; }
```

### Data

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using *LodInput.GetData* which will validate and cast.

*LodInputModule*.Global and *LodInputModule*.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg *LodInputModule.Texture* mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

### Declaration

```
public LodInputData Data { get; }
```

## 1.94.3 Inherited Methods

### GetData

#### GetData<T>()

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

### Declaration

```
public T GetData<T>() where T : LodInputData
```

### Return

The casted data.

## 1.95 ShapeWaves

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *LodInput*

Base class for Shape components.

### 1.95.1 Properties

#### EvaluateSpectrumAtRunTimeEveryFrame

Whether to evaluate the spectrum every frame.

When false, the wave spectrum is evaluated once on startup in editor play mode and standalone builds, rather than every frame. This is less flexible, but it reduces the performance cost significantly.

#### Declaration

```
public bool EvaluateSpectrumAtRunTimeEveryFrame { get; set; }
```

#### IncludeInDropDetailHeightBasedOnWaves

Whether the maximum possible vertical displacement is used for the Drop Detail Height Based On Waves calculation.

This setting is ignored for global waves, as they always contribute. For local waves, only enable for large areas that are treated like global waves (eg a storm).

#### Declaration

```
public bool IncludeInDropDetailHeightBasedOnWaves { get; set; }
```

#### OverrideGlobalWindDirection

Whether to use the wind direction on this component rather than the global wind direction.

Global wind direction comes from the Water Renderer component.

#### Declaration

```
public bool OverrideGlobalWindDirection { get; set; }
```

#### OverrideGlobalWindSpeed

Whether to use the wind speed on this component rather than the global wind speed.

Global wind speed comes from the Water Renderer component.

#### Declaration

```
public bool OverrideGlobalWindSpeed { get; set; }
```

#### Resolution

Resolution to use for wave generation buffers.

Low resolutions are more efficient but can result in noticeable patterns in the shape.

#### Declaration

```
public int Resolution { get; set; }
```

## RespectShallowWaterAttenuation

How much these waves respect the shallow water attenuation.

Attenuation is defined on the Animated Waves. Set to zero to ignore attenuation.

### Declaration

```
public float RespectShallowWaterAttenuation { get; set; }
```

## SeaLevelOnly

Whether global waves is applied above or below sea level.

Waves are faded out to avoid hard transitions. They are fully faded by 1m from sea level.

### Declaration

```
public bool SeaLevelOnly { get; set; }
```

## Spectrum

The spectrum that defines the water surface shape.

### Declaration

```
public WaveSpectrum Spectrum { get; set; }
```

## WaveDirectionHeadingAngle

Primary wave direction heading (degrees).

This is the angle from x axis in degrees that the waves are oriented towards. If a spline is being used to place the waves, this angle is relative to the spline.

### Declaration

```
public float WaveDirectionHeadingAngle { get; set; }
```

## WindSpeed

Wind speed in km/h. Controls wave conditions.

### Declaration

```
public float WindSpeed { get; set; }
```

## WindSpeedKPH

The wind speed in kilometers per hour (KPH).

Wind speed can come from this component or the [WaterRenderer](#).

**Declaration**

```
public float WindSpeedKPH { get; }
```

**WindSpeedMPS**

The wind speed in meters per second (MPS).

Wind speed can come from this component or the *WaterRenderer*.

**Declaration**

```
public float WindSpeedMPS { get; }
```

**1.95.2 Inherited Properties****Blend**

How this input blends into existing data.

Similar to blend operations in shaders. For inputs which have materials, use the blend functionality on the shader/material.

**Declaration**

```
public LodInputBlend Blend { get; set; }
```

**FeatherWidth**

The width of the feathering to soften the edges to blend inputs.

Inputs that do not support feathering will have this field disabled or hidden in UI.

**Declaration**

```
public float FeatherWidth { get; set; }
```

**FollowHorizontalWaveMotion**

How this input responds to horizontal displacement.

If false, data will not move horizontally with the waves. Has a small performance overhead when disabled. Only suitable for inputs of small size.

**Declaration**

```
public bool FollowHorizontalWaveMotion { get; set; }
```

**Mode**

The mode for this input.

See the manual for more details about input modes. Use AddComponent(LodInputModule) to set the mode via scripting. The mode cannot be changed after creation.

**Declaration**

```
public LodInputMode Mode { get; }
```

**Queue**

The order this input will render.

Order is Queue plus SiblingIndex

**Declaration**

```
public int Queue { get; set; }
```

**Weight**

Scales the input.

**Declaration**

```
public float Weight { get; set; }
```

**ForceRenderingOff**

Disables rendering of input into data, but continues most scripting activities.

**Declaration**

```
public bool ForceRenderingOff { get; set; }
```

**Data**

Properties specific to LodInput.Mode.

You will need to cast to a more specific type to change certain properties. Types derive from and end with *LodInputData*. Consider using LodInput.GetData which will validate and cast.

LodInputModule.Global and LodInputModule.Primitive will have no associated data and will be null. The rest will have an *LodInputData* type which will be prefixed with the input type and then mode (eg LodInputModule.Texture mode for *FoamLodInput* will be *FoamTextureLodInputData* ).

An exception is *ShapeGerstner* and *ShapeFFT*. They are derived from *ShapeWaves* and use it as a prefix. (eg *ShapeWavesTextureLodInputData* ).

**Declaration**

```
public LodInputData Data { get; }
```

**1.95.3 Inherited Methods****GetData**

**GetData<T>()**

Retrieves the typed data and validates the passed type.

Validation is stripped from builds.

**Declaration**

```
public T GetData<T>() where T : LodInputData
```

**Return**

The casted data.

## 1.96 ShapeWavesRendererLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *RendererLodInputData*

Data storage for for the Renderer input mode.

### 1.96.1 Inherited Properties

**CheckShaderName**

Check that the shader applied to this object matches the input type.

For example, an Animated Waves input object has an Animated Waves input shader.

**Declaration**

```
public bool CheckShaderName { get; set; }
```

**CheckShaderPasses**

Check that the shader applied to this object has only a single pass, as only the first pass is executed for most inputs.

**Declaration**

```
public bool CheckShaderPasses { get; set; }
```

**DisableRenderer**

Forces the renderer to only render into the LOD data, and not to render in the scene as it normally would.

**Declaration**

```
public bool DisableRenderer { get; set; }
```

**OverrideShaderPass**

Whether to set the shader pass manually.

**Declaration**

```
public bool OverrideShaderPass { get; set; }
```

**Renderer**

The renderer to use for this input.

Can be anything that inherits from *Renderer* like *MeshRenderer* , *TrailRenderer* etc.

**Declaration**

```
public Renderer Renderer { get; set; }
```

**ShaderPassIndex**

The shader pass to execute.

Set to -1 to execute all passes.

**Declaration**

```
public int ShaderPassIndex { get; set; }
```

## 1.97 ShapeWavesTextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *DirectionalTextureLodInputData*

Data storage for for the Texture input mode.

### 1.97.1 Inherited Properties

**NegativeValues**

Whether the texture supports negative values.

**Declaration**

```
public bool NegativeValues { get; set; }
```

**Multiplier**

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

**Texture**

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.98 ShorelineVolumeColorSource

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

The source of depth color.

### 1.98.1 Properties

**None**

No depth color.

**Declaration**

```
None = 0
```

**Depth**

Depth color based on water depth.

**Declaration**

```
Depth = 1
```

**Distance**

Depth color based on shoreline distance.

**Declaration**

```
Distance = 2
```

## 1.99 SphereWaterInteraction

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Approximates the interaction between a sphere and the water.

Multiple spheres can be used to model the interaction of a non-spherical shape.

### 1.99.1 Properties

#### BoostLargeWaves

Whether to improve visibility in larger LODs.

If the dynamic waves are not visible far enough in the distance from the camera, this can be used to boost the output.

#### Declaration

```
public bool BoostLargeWaves { get; set; }
```

#### CompensateForWaveMotion

How much to correct the position for horizontal wave displacement.

If set to 0, the input will always be applied at a fixed position before any horizontal displacement from waves. If waves are large then their displacement may cause the interactive waves to drift away from the object. This parameter can be increased to compensate for this displacement and combat this issue. However increasing too far can cause a feedback loop which causes strong ‘ring’ artifacts to appear in the dynamic waves. This parameter can be tweaked to balance this two effects.

#### Declaration

```
public float CompensateForWaveMotion { get; set; }
```

#### InnerSphereMultiplier

Model parameter that can be used to modify the shape of the interaction.

Internally the interaction is modelled by a pair of nested spheres. The forces from the two spheres combine to create the final effect. This parameter scales the effect of the inner sphere and can be tweaked to adjust the shape of the result.

#### Declaration

```
public float InnerSphereMultiplier { get; set; }
```

#### InnerSphereOffset

Model parameter that can be used to modify the shape of the interaction.

This parameter controls the size of the inner sphere and can be tweaked to give further control over the result.

#### Declaration

```
public float InnerSphereOffset { get; set; }
```

### MaximumSpeed

Maximum speed clamp (km/h).

Useful for controlling/limiting wake.

#### Declaration

```
public float MaximumSpeed { get; set; }
```

### Radius

Radius of the sphere that is modelled from which the interaction forces are calculated.

#### Declaration

```
public float Radius { get; set; }
```

### TeleportSpeed

Teleport speed (km/h).

If the calculated speed is larger than this amount, the object is deemed to have teleported and the computed velocity is discarded.

#### Declaration

```
public float TeleportSpeed { get; set; }
```

### VelocityOffset

Offset in direction of motion to help ripples appear in front of sphere.

There is some latency between applying a force to the wave simulation and the resulting waves appearing. Applying this offset can help to ensure the waves do not lag behind the sphere.

#### Declaration

```
public float VelocityOffset { get; set; }
```

### WarnOnSpeedClamp

Outputs a warning to the console on speed clamp.

#### Declaration

```
public bool WarnOnSpeedClamp { get; set; }
```

### WarnOnTeleport

Outputs a warning to the console on teleport.

**Declaration**

```
public bool WarnOnTeleport { get; set; }
```

**Weight**

Intensity of the forces.

Can be set negative to invert.

**Declaration**

```
public float Weight { get; set; }
```

**WeightVerticalMultiplier**

Intensity of the forces from vertical motion of the sphere.

Scales ripples generated from a sphere moving up or down.

**Declaration**

```
public float WeightVerticalMultiplier { get; set; }
```

## 1.100 SurfaceRenderer

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Renders the water surface.

### 1.100.1 Properties

**AllowRenderQueueSorting**

Whether to allow sorting using the render queue.

If you need to change the minor part of the render queue (eg +100), then enable this option. As a side effect, it will also disable the front-to-back rendering optimization for Crest. This option does not affect changing the major part of the render queue (eg AlphaTest, Transparent), as that is always allowed.

**Declaration**

```
public bool AllowRenderQueueSorting { get; set; }
```

**CameraExclusions**

Rules to exclude cameras from surface rendering.

These are exclusion rules, so for all cameras, select Nothing. These rules are applied on top of the Layer rules.

## Declaration

```
public WaterCameraExclusion CameraExclusions { get; set; }
```

## CastShadows

Have the water surface cast shadows for albedo (both foam and custom).

## Declaration

```
public bool CastShadows { get; set; }
```

## Enabled

Whether the underwater effect is enabled.

Allocates/releases resources if state has changed.

## Declaration

```
public bool Enabled { get; set; }
```

## Layer

The water chunk renderers will have this layer.

## Declaration

```
public int Layer { get; set; }
```

## Material

Material to use for the water surface.

## Declaration

```
public Material Material { get; set; }
```

## TimeSliceBoundsUpdateFrameCount

How many frames to distribute the chunk bounds calculation.

The chunk bounds are calculated per frame to ensure culling is correct when using inputs that affect displacement. Some performance can be saved by distributing the load over several frames. The higher the frames, the longer it will take - lowest being instant.

## Declaration

```
public int TimeSliceBoundsUpdateFrameCount { get; set; }
```

## VolumeMaterial

Underwater will copy from this material if set.

Useful for overriding properties for the underwater effect. To see what properties can be overriden, see the disabled properties on the underwater material. This does not affect the surface.

### Declaration

```
public Material VolumeMaterial { get; set; }
```

## WaterBodyCulling

Whether ‘Water Body’ components will cull the water tiles.

Disable if you want to use the ‘Material Override’ feature and still have an ocean.

### Declaration

```
public bool WaterBodyCulling { get; set; }
```

## 1.100.2 Events

### OnCreateChunkRenderer

Invoked after water chunk modification.

Gives an opportunity to modify the renderer.

### Declaration

```
public static Action<Renderer> OnCreateChunkRenderer { get; set; }
```

## 1.101 TextureLodInputData

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Extends *LodInputData*

Data storage for for the Texture input mode.

### 1.101.1 Properties

#### Multiplier

Multiplies the texture sample.

This is useful for normalized textures. The four components map to the four color/alpha components of the texture (if they exist).

**Declaration**

```
public Vector4 Multiplier { get; set; }
```

**Texture**

Texture to render into the simulation.

**Declaration**

```
public Texture Texture { get; set; }
```

## 1.102 TimeProvider

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Extends [ITimeProvider](#)

A behaviour which is driven by a ManagerBehaviour instead of Unity's event system.

### 1.102.1 Properties

**Time**

Current time.

**Declaration**

```
public abstract float Time { get; }
```

**Delta**

Delta time.

**Declaration**

```
public abstract float Delta { get; }
```

## 1.103 UnderwaterRenderer

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Renders the underwater effect.

### 1.103.1 Properties

## AllCameras

### Obsolete

Please use Camera Exclusion instead.

Whether to execute for all cameras.

If disabled, then additionally ignore any camera that is not the view camera or our reflection camera. It will require managing culling masks of all cameras.

### Declaration

```
public bool AllCameras { get; set; }
```

## CameraExclusions

Rules to exclude cameras from rendering underwater.

These are exclusion rules, so for all cameras, select Nothing. These rules are applied on top of the Layer rules.

### Declaration

```
public WaterCameraExclusion CameraExclusions { get; set; }
```

## CopyWaterMaterialParametersEachFrame

Copying parameters each frame ensures underwater appearance stays consistent with the water surface.

Has a small overhead so should be disabled if not needed.

### Declaration

```
public bool CopyWaterMaterialParametersEachFrame { get; set; }
```

## CullLimit

Proportion of visibility below which the water surface will be culled when underwater.

The larger the number, the closer to the camera the water tiles will be culled.

### Declaration

```
public float CullLimit { get; set; }
```

## Enabled

Whether the underwater effect is enabled.

Allocates/releases resources if state has changed.

## Declaration

```
public bool Enabled { get; set; }
```

## AffectsEnvironmentalLighting

Provides out-scattering based on the camera's underwater depth.

It scales down environmental lighting (sun, reflections, ambient etc) with the underwater depth. This works with vanilla lighting, but uncommon or custom lighting will require a custom solution (use this for reference)

## Declaration

```
public bool AffectsEnvironmentalLighting { get; set; }
```

## EnvironmentalLightingWeight

How much this effect applies.

Values less than 1 attenuate light less underwater. Value of 1 is physically based.

## Declaration

```
public float EnvironmentalLightingWeight { get; set; }
```

## FarPlaneMultiplier

Adjusts the far plane for horizon line calculation. Helps with horizon line issue.

## Declaration

```
public float FarPlaneMultiplier { get; set; }
```

## Layer

Any camera or probe with this layer in its culling mask will render underwater.

## Declaration

```
public int Layer { get; set; }
```

## Material

The underwater material. The water surface material is copied into this material.

## Declaration

```
public Material Material { get; set; }
```

## 1.103.2 Static Properties

### AfterCopyMaterial

Raised after copying the water material properties to the underwater material.

#### Declaration

```
public static Action<WaterRenderer, Material> AfterCopyMaterial { get; set; }
```

## 1.104 WaterBody

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

Demarcates an AABB area where water is present in the world.

If present, water tiles will be culled if they don't overlap any WaterBody.

### 1.104.1 Properties

#### BelowSurfaceMaterial

Overrides the property on the Water Renderer with the same name when the camera is inside the bounds.

#### Declaration

```
public Material BelowSurfaceMaterial { get; set; }
```

#### Clipped

Makes sure this water body is not clipped.

If clipping is enabled and set to clip everywhere by default, this option will register this water body to ensure its area does not get clipped.

#### Declaration

```
public bool Clipped { get; set; }
```

#### AboveSurfaceMaterial

Water chunks that overlap this waterbody area will be assigned this material.

This is useful for varying water appearance across different water bodies. If no override material is specified, the default material assigned to the WaterRenderer component will be used.

#### Declaration

```
public Material AboveSurfaceMaterial { get; set; }
```

## VolumeMaterial

Overrides the Water Renderer's volume material when the camera is inside the bounds.

### Declaration

```
public Material VolumeMaterial { get; set; }
```

## 1.105 WaterCamera

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Marks a camera to render water regardless of exclusions.

This is only necessary when using [WaterCameraExclusion](#).

## 1.106 WaterCameraExclusion

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Rules to exclude cameras.

### 1.106.1 Properties

#### Nothing

No exclusion rules applied.

### Declaration

```
Nothing = 0
```

#### Hidden

Exclude hidden cameras.

Does not affect reflection cameras, as they are typically always hidden. Use the Reflection flag for them.

### Declaration

```
Hidden = 2
```

#### Reflection

Exclude reflection cameras.

### Declaration

```
Reflection = 4
```

## NonMainCamera

Exclude cameras not tagged as MainCamera.

### Declaration

```
NonMainCamera = 8
```

## Everything

Apply all exclusion rules.

### Declaration

```
Everything = -1
```

## 1.107 WaterDataBackgroundMode

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

The background processing mode for when cameras do not render.

### 1.107.1 Properties

#### Always

Always progress simulations in the background when camera does not render.

### Declaration

```
Always = 0
```

#### Inactive

Progress simulations in the background when camera is inactive (ie !isActiveAndEnabled).

### Declaration

```
Inactive = 1
```

#### Disabled

Progress simulations in the background when camera is disabled (ie !enabled).

### Declaration

```
Disabled = 2
```

**Never**

Never progress simulations in the background.

**Declaration**

```
Never = 3
```

## 1.108 WaterInjectionPoint

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

The render pass injection point.

### 1.108.1 Properties

**Default**

Renders in the default pass.

For the water surface, this will be determined by the material (opaque or transparent). This pass is controlled by Unity, and is not compatible with certain features like soft particles.

**Declaration**

```
Default = 0
```

**BeforeTransparent**

Renders before the transparent pass.

This has advantages like being compatible with soft particles, refractive shaders, and possibly third-party fog.

**Declaration**

```
BeforeTransparent = 1
```

## 1.109 WaterReflectionSide

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

What side of the water surface to render planar reflections for.

### 1.109.1 Properties

**Both**

Both sides. Most expensive.

## Declaration

```
Both = 0
```

### Above

Above only. Typical for planar reflections.

## Declaration

```
Above = 1
```

### Below

Below only. For total internal reflections.

## Declaration

```
Below = 2
```

# 1.110 WaterReflections

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Renders reflections for water. Currently on planar reflections.

## 1.110.1 Properties

### AllowMSAA

#### Obsolete

MSAA for the planar reflection camera is no longer supported. This setting will be ignored.

Whether to allow MSAA.

## Declaration

```
public bool AllowMSAA { get; set; }
```

### ClipPlaneOffset

The near clip plane clips any geometry before it, removing it from reflections.

Can be used to reduce reflection leaks and support varied water level.

**Declaration**

```
public float ClipPlaneOffset { get; set; }
```

**DisableOcclusionCulling**

Disables occlusion culling.

**Declaration**

```
public bool DisableOcclusionCulling { get; set; }
```

**DisablePixelLights**

Disables pixel lights (BIRP only).

**Declaration**

```
public bool DisablePixelLights { get; set; }
```

**DisableShadows**

Disables shadows.

**Declaration**

```
public bool DisableShadows { get; set; }
```

**Enabled**

Whether planar reflections are enabled.

Allocates/releases resources if state has changed.

**Declaration**

```
public bool Enabled { get; set; }
```

**FarClipPlane**

Anything beyond the far clip plane is not rendered.

**Declaration**

```
public float FarClipPlane { get; set; }
```

**HDR**

Whether to allow HDR.

## Declaration

```
public bool HDR { get; set; }
```

## Layers

The layers to render into reflections.

## Declaration

```
public LayerMask Layers { get; set; }
```

## ReflectionSide

What side of the water surface to render planar reflections for.

## Declaration

```
public WaterReflectionSide ReflectionSide { get; set; }
```

## NonObliqueNearSurface

Planar reflections using an oblique frustum for better performance.

This can cause depth issues for TIRs, especially near the surface.

## Declaration

```
public bool NonObliqueNearSurface { get; set; }
```

## NonObliqueNearSurfaceThreshold

If within this distance from the surface, disable the oblique matrix.

## Declaration

```
public float NonObliqueNearSurfaceThreshold { get; set; }
```

## Overscan

Overscan amount to capture off-screen content.

Renders the reflections at a larger viewport size to capture off-screen content when the surface reflects off-screen. This avoids a category of artifacts - especially when looking down. This can be expensive, as the value is a multiplier to the capture size.

## Declaration

```
public float Overscan { get; set; }
```

## QualitySettingsOverride

Overrides global quality settings.

### Declaration

```
public QualitySettingsOverride QualitySettingsOverride { get; set; }
```

## RenderOnlySingleCamera

Whether to render to the viewer camera only.

When disabled, reflections will render for all cameras rendering the water layer, which currently this prevents Refresh Rate from working. Enabling will unlock the Refresh Rate heading.

### Declaration

```
public bool RenderOnlySingleCamera { get; set; }
```

## Resolution

Resolution of the reflection texture.

### Declaration

```
public int Resolution { get; set; }
```

## Sky

Whether to render the sky or fallback to default reflections.

Not rendering the sky can prevent other custom shaders (like tree leaves) from being in the final output. Enable for best compatibility.

### Declaration

```
public bool Sky { get; set; }
```

## Stencil

Whether to allow stencil operations.

### Declaration

```
public bool Stencil { get; set; }
```

## UseObliqueMatrix

An oblique matrix will clip anything below the surface for free.

Disable if you have problems with certain effects. Disabling can cause other artifacts like objects below the surface to appear in reflections.

## Declaration

```
public bool UseObliqueMatrix { get; set; }
```

## Mode

What side of the water surface to render planar reflections for.

## Declaration

```
public WaterReflectionSide Mode { get; set; }
```

## 1.110.2 Events

### OnCameraAdded

Invoked when the reflection camera is created.

## Declaration

```
public static Action<Camera> OnCameraAdded { get; set; }
```

## 1.111 WaterRenderer

Class in *WaveHarmonic.Crest*, WaveHarmonic.Crest

The main script for the water system.

Attach this to an object to create water. This script initializes the various data types and systems and moves/scales the water based on the viewpoint. It also hosts a number of global settings that can be tweaked here.

## 1.111.1 Properties

### ViewerHeightAboveWater

Vertical offset of camera vs water surface.

## Declaration

```
public float ViewerHeightAboveWater { get; }
```

### ViewpointHeightAboveWater

Vertical offset of viewpoint vs water surface.

## Declaration

```
public float ViewpointHeightAboveWater { get; }
```

## ViewerDistanceToShoreline

Distance of camera to shoreline. Positive if over water and negative if over land.

### Declaration

```
public float ViewerDistanceToShoreline { get; }
```

## SeaLevel

Sea level is given by y coordinate of GameObject with WaterRenderer script.

### Declaration

```
public float SeaLevel { get; }
```

## TimeProviders

Loosely a stack for time providers.

The last WaterRenderer.TimeProvider in the list is the active one. When a WaterRenderer.TimeProvider gets added to the stack, it is bumped to the top of the list. When a WaterRenderer.TimeProvider is removed, all instances of it are removed from the stack. This is less rigid than a real stack which would be harder to use as users have to keep a close eye on the order that things are pushed/popped.

### Declaration

```
public Stack<ITimeProvider> TimeProviders { get; }
```

## TimeProvider

The current time provider.

### Declaration

```
public ITimeProvider TimeProvider { get; }
```

## Gravity

Physics gravity applied to water.

### Declaration

```
public float Gravity { get; }
```

## Scale

Current water scale (changes with viewer altitude).

## Declaration

```
public float Scale { get; }
```

## CollisionProvider

Provides water shape to CPU.

## Declaration

```
public ICollisionProvider CollisionProvider { get; }
```

## FlowProvider

Provides flow to the CPU.

## Declaration

```
public IFlowProvider FlowProvider { get; }
```

## DepthProvider

Provides water depth and distance to water edge to the CPU.

## Declaration

```
public IDepthProvider DepthProvider { get; }
```

## AbsorptionLod

Absorption information - gives color to water.

## Declaration

```
public AbsorptionLod AbsorptionLod { get; }
```

## AlbedoLod

Albedo - a colour layer composited onto the water surface.

## Declaration

```
public AlbedoLod AlbedoLod { get; }
```

## AllowRenderQueueSorting

### ⓘ Obsolete

This property can now be found on WaterRenderer.Surface

Whether to allow sorting using the render queue.

If you need to change the minor part of the render queue (eg +100), then enable this option. As a side effect, it will also disable the front-to-back rendering optimization for Crest. This option does not affect changing the major part of the render queue (eg AlphaTest, Transparent), as that is always allowed.

### Declaration

```
public bool AllowRenderQueueSorting { get; set; }
```

### AnimatedWavesLod

All waves (including Dynamic Waves) are written to this simulation.

### Declaration

```
public AnimatedWavesLod AnimatedWavesLod { get; }
```

### Viewer

The camera which drives the water data.

Setting this is optional. Defaults to the main camera.

### Declaration

```
public Camera Viewer { get; set; }
```

### CameraExclusions

Rules to exclude cameras from being a center-of-detail.

These are exclusion rules, so for all cameras, select Nothing.

### Declaration

```
public WaterCameraExclusion CameraExclusions { get; set; }
```

### CastShadows

#### Obsolete

This property can now be found on WaterRenderer.Surface

Have the water surface cast shadows for albedo (both foam and custom).

### Declaration

```
public bool CastShadows { get; set; }
```

### CenterOfDetailDisplacementCorrection

Keep the center of detail from drifting from the viewpoint.

Large horizontal displacement can displace the center of detail. This uses queries to keep the center of detail aligned.

#### Declaration

```
public bool CenterOfDetailDisplacementCorrection { get; set; }
```

### ClipLod

Clip surface information for clipping the water surface.

#### Declaration

```
public ClipLod ClipLod { get; }
```

### DataBackgroundMode

The background rendering mode when a camera does not render.

When switching between multiple cameras, simulations will not progress for cameras which do not render for that frame. This setting tells the system when it should continue to progress the simulations.

#### Declaration

```
public WaterDataBackgroundMode DataBackgroundMode { get; set; }
```

### DepthLod

Water depth information used for shallow water, shoreline foam, wave attenuation, among others.

#### Declaration

```
public DepthLod DepthLod { get; }
```

### DropDetailHeightBasedOnWaves

Drops the height for maximum water detail based on waves.

This means if there are big waves, max detail level is reached at a lower height, which can help visual range when there are very large waves and camera is at sea level.

#### Declaration

```
public float DropDetailHeightBasedOnWaves { get; set; }
```

### DynamicWavesLod

Dynamic waves generated from interactions with objects such as boats.

**Declaration**

```
public DynamicWavesLod DynamicWavesLod { get; }
```

**ExtentsSizeMultiplier**

Applied to the extents' far vertices to make them larger.

Increase if the extents do not reach the horizon or you see the underwater effect at the horizon.

**Declaration**

```
public float ExtentsSizeMultiplier { get; set; }
```

**FlowLod**

Horizontal motion of water body, akin to water currents.

**Declaration**

```
public FlowLod FlowLod { get; }
```

**FoamLod**

Simulation of foam created in choppy water and dissipating over time.

**Declaration**

```
public FoamLod FoamLod { get; }
```

**ForceScaleChangeSmoothing**

Forces smoothing for scale changes.

When water level varies, smoothing scale change can prevent pops when the viewer's height above water sharply changes. Smoothing is disabled when terrain sampling is enabled or the water level simulation is disabled.

**Declaration**

```
public bool ForceScaleChangeSmoothing { get; set; }
```

**GeometryDownSampleFactor**

How much of the water shape gets tessellated by geometry.

For example, if set to four, every geometry quad will span 4x4 LOD data texels. a value of 2 will generate one vert per 2x2 LOD data texels. A value of 1 means a vert is generated for every LOD data texel. Larger values give lower fidelity surface shape with higher performance.

## Declaration

```
public int GeometryDownSampleFactor { get; set; }
```

## GravityMultiplier

Multiplier for physics gravity.

## Declaration

```
public float GravityMultiplier { get; set; }
```

## GravityOverride

Gravity for all wave calculations.

## Declaration

```
public float GravityOverride { get; set; }
```

## InjectionPoint

When in the render pipeline the water is rendered.

Default is the old behaviour which is controlled by Unity.

## Declaration

```
public WaterInjectionPoint InjectionPoint { get; set; }
```

## Layer

### Obsolete

This property can now be found on WaterRenderer.Surface

The water chunk renderers will have this layer.

## Declaration

```
public int Layer { get; set; }
```

## LevelLod

Varying water level to support water bodies at different heights and rivers to run down slopes.

## Declaration

```
public LevelLod LevelLod { get; }
```

## Material

### Obsolete

This property can now be found on WaterRenderer.Surface

Material to use for the water surface.

### Declaration

```
public Material Material { get; set; }
```

## Meniscus

The meniscus module.

### Declaration

```
public Meniscus Meniscus { get; }
```

## OverrideGravity

Provide your own gravity value instead of Physics.gravity.

### Declaration

```
public bool OverrideGravity { get; set; }
```

## OverrideRenderHDR

Whether to override the automatic detection of framebuffer HDR rendering (BIRP only).

Rendering using HDR formats is optional, but there is no way for us to determine if HDR rendering is enabled in the Graphics Settings. We make an educated guess based on which platform is the target. If you see rendering issues, try disabling this.

### Declaration

```
public bool OverrideRenderHDR { get; set; }
```

## Portals

The portal renderer.

### Declaration

```
public PortalRenderer Portals { get; }
```

## PrimaryLight

The primary light that affects the water.

Setting this is optional. This should be a directional light. Defaults to RenderSettings.sun.

### Declaration

```
public Light PrimaryLight { get; set; }
```

## Reflections

The reflection renderer.

### Declaration

```
public WaterReflections Reflections { get; }
```

## RenderHDR

Force HDR format usage (BIRP only).

If enabled, we assume the framebuffer is an HDR format, otherwise an LDR format.

### Declaration

```
public bool RenderHDR { get; set; }
```

## LodResolution

The resolution of the various water LOD data.

This includes mesh density, displacement textures, foam data, dynamic wave simulation, etc. Sets the ‘detail’ present in the water - larger values give more detail at increased run-time expense. This value can be overriden per LOD in their respective settings except for Animated Waves which is tied to this value.

### Declaration

```
public int LodResolution { get; set; }
```

## SampleTerrainHeightForScale

Also checks terrain height when determining the scale.

The scale is changed based on the viewer’s height above the water surface. This can be a problem with varied water level, as the viewer may not be directly over the higher water level leading to a height difference, and thus incorrect scale.

### Declaration

```
public bool SampleTerrainHeightForScale { get; set; }
```

## ScaleRange

The scale the water can be (infinity for no maximum).

Water is scaled horizontally with viewer height, to keep the meshing suitable for elevated viewpoints. This sets the minimum and maximum the water will be scaled. Low minimum values give lots of detail, but will limit the horizontal extents of the water detail. Increasing the minimum value can be a great performance saving for mobile as it will reduce draw calls.

### Declaration

```
public Vector2 ScaleRange { get; set; }
```

## ScatteringLod

Scattering information - gives color to water.

### Declaration

```
public ScatteringLod ScatteringLod { get; }
```

## ShadowLod

Shadow information used for lighting water.

### Declaration

```
public ShadowLod ShadowLod { get; }
```

## LodLevels

Number of levels of details (chunks, scales etc) to generate.

The horizontal range of the water surface doubles for each added LOD, while GPU processing time increases linearly. The higher the number, the further out detail will be. Furthermore, the higher the count, the more larger wavelengths can be filtering in queries.

### Declaration

```
public int LodLevels { get; set; }
```

## Surface

The water surface renderer.

### Declaration

```
public SurfaceRenderer Surface { get; }
```

## TeleportThreshold

The distance threshold for when the viewer has considered to have teleported.

This is used to prevent popping, and for prewarming simulations. Threshold is in Unity units.

### Declaration

```
public float TeleportThreshold { get; set; }
```

## TimeSliceBoundsUpdateFrameCount

### Obsolete

This property can now be found on WaterRenderer.Surface

How many frames to distribute the chunk bounds calculation.

The chunk bounds are calculated per frame to ensure culling is correct when using inputs that affect displacement. Some performance can be saved by distributing the load over several frames. The higher the frames, the longer it will take - lowest being instant.

### Declaration

```
public int TimeSliceBoundsUpdateFrameCount { get; set; }
```

## Underwater

The underwater renderer.

### Declaration

```
public UnderwaterRenderer Underwater { get; }
```

## Viewpoint

The viewpoint which drives the water detail - the center of the LOD system.

Setting this is optional. Defaults to the camera.

### Declaration

```
public Transform Viewpoint { get; set; }
```

## VolumeMaterial

### Obsolete

This property can now be found on WaterRenderer.Surface

Underwater will copy from this material if set.

Useful for overriding properties for the underwater effect. To see what properties can be overriden, see the disabled properties on the underwater material. This does not affect the surface.

### Declaration

```
public Material VolumeMaterial { get; set; }
```

### WaterBodyCulling

#### Obsolete

This property can now be found on WaterRenderer.Surface

Whether ‘Water Body’ components will cull the water tiles.

Disable if you want to use the ‘Material Override’ feature and still have an ocean.

### Declaration

```
public bool WaterBodyCulling { get; set; }
```

### WindDirection

Base wind direction in degrees.

Controls wave conditions. Can be overridden on Shape\* components.

### Declaration

```
public float WindDirection { get; set; }
```

### WindSpeed

Base wind speed in km/h.

Controls wave conditions. Can be overridden on Shape\* components.

### Declaration

```
public float WindSpeed { get; set; }
```

### WindTurbulence

Base wind turbulence.

Controls wave conditions. Can be overridden on ShapeFFT components.

## Declaration

```
public float WindTurbulence { get; set; }
```

## WindZone

Uses a provided WindZone as the source of global wind.

It must be directional. Wind speed units are presumed to be in m/s.

## Declaration

```
public WindZone WindZone { get; set; }
```

## WriteMotionVectors

Whether to enable motion vector support.

## Declaration

```
public bool WriteMotionVectors { get; set; }
```

## WriteToColorTexture

Whether to write the water surface color to the color/opaque texture.

This is likely only beneficial if the water injection point is before transparency, and there are shaders which need it (like refraction).

## Declaration

```
public bool WriteToColorTexture { get; set; }
```

## WriteToDepthTexture

Whether to write the water surface depth to the depth texture.

The water surface writes to the depth buffer, but Unity does not copy it to the depth texture for post-processing effects like Depth of Field (or refraction). This will copy the depth buffer to the depth texture.

## Declaration

```
public bool WriteToDepthTexture { get; set; }
```

## 1.111.2 Static Properties

### RunningWithoutGraphics

Is runtime environment without graphics card

## Declaration

```
public static bool RunningWithoutGraphics { get; }
```

## RunningHeadless

### Obsolete

We no longer care whether Unity is running in non-interactive mode.

Is runtime environment non-interactive (not displaying to user).

## Declaration

```
public static bool RunningHeadless { get; }
```

## FrameCount

The frame count for Crest.

## Declaration

```
public static int FrameCount { get; }
```

## 1.111.3 Static Methods

### CalculateAbsorptionValueFromColor

#### CalculateAbsorptionValueFromColor(Color)

Calculates the absorption value from the absorption color.

## Declaration

```
public static Vector4 CalculateAbsorptionValueFromColor(Color color)
```

## Parameters

color	The absorption color.
-------	-----------------------

## Return

The absorption value (XYZ value).

## 1.112 WatertightHull

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Removes water from a provided hull using the clip simulation.

### 1.112.1 Properties

#### Inverted

Inverts the effect to remove clipping (ie add water).

#### Declaration

```
public bool Inverted { get; set; }
```

#### Mesh

The convex hull to keep water out.

#### Declaration

```
public Mesh Mesh { get; set; }
```

#### Mode

Which mode to use.

#### Declaration

```
public WatertightHullMode Mode { get; set; }
```

#### Queue

Order this input will render.

Queue is ‘Queue + SiblingIndex’

#### Declaration

```
public int Queue { get; set; }
```

#### UseClipWithDisplacement

Whether to also to clip the surface when using displacement mode.

Displacement mode can have a leaky hull by allowing chop top push waves across the hull boundaries slightly. Clipping the surface will remove these interior leaks.

#### Declaration

```
public bool UseClipWithDisplacement { get; set; }
```

## 1.113 WatertightHullMode

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

The mode for *WatertightHull*.

Each mode has its strengths and weaknesses.

### 1.113.1 Properties

#### Displacement

Use displacement to remove water.

Using displacement will also affect the underwater and can nest buoyant objects. Requires the displacement layer to be enabled.

#### Declaration

```
Displacement = 0
```

#### Clip

Clips the surface to remove water.

This option is more precise and can be submerged.

#### Declaration

```
Clip = 1
```

## 1.114 WaveSampling

Enum in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

The wave sampling method to determine quality and performance.

### 1.114.1 Properties

#### Automatic

Automatically chooses the other options as needed (512+ resolution needs precision).

#### Declaration

```
Automatic = 0
```

## Performance

Reduces samples by copying waves from higher LODs to lower LODs.

Best for resolutions lower than 512.

## Declaration

```
Performance = 1
```

## Precision

Samples directly from the wave buffers to preserve wave quality.

Needed for higher resolutions (512+). Higher LOD counts can also benefit with this enabled.

## Declaration

```
Precision = 2
```

# 1.115 WaveSpectrum

Class in [WaveHarmonic.Crest](#), WaveHarmonic.Crest

Water shape representation - power values for each octave of wave components.