



**REAL-TIME  
WEATHER**

**PRO**

1.2.0

Documentation

**Thank you for your purchase!**

**ASSIST**  
Innovative Minds

## OVERVIEW & INSTALLATION

Installation and Quick Start	3
Enviro Integration	10
Tenkoku Integration	11
Atmos Integration	12
Expanse Integration	14
Crest Ocean Integration	15
KWS Integration	16
Forecast Timelapse	17
Help and Bug Report	21
Feedback Form	22
FAQs	23

## SCRIPTING API

Real-Time Weather Manager	25
Weather Data	26
Atlas Module	29
Weather For You Module	30
Weather Underground Module	31
Reserve Locations	33
Open Weather Map Service	34
Tomorrow.io Service	40
MetOcean Service	50
Stormglass Service	56
Reverse Geocoding	64
Custom Forecast Window	66
Platform Compatibility	73
URP/HDRP Compatibility	74

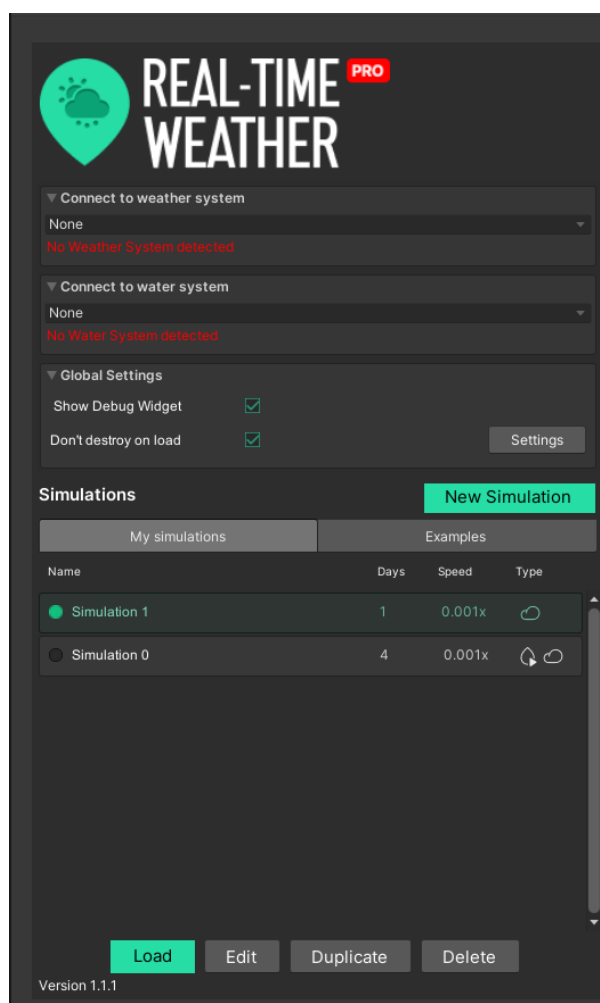
# Overview & Installation

## Installation

Import the Real-Time Weather package and follow these steps to get started:

### STEP 1

On the Unity top bar, you will find a new tab “Real-Time Weather”, so click on it and then select “Real-Time Weather Manager”. A new GameObject will be created in your scene with the “RealTimeWeatherManager” component.



## STEP 2

---

To see an overlay with info about the data received from providers, check "Show Debug Widget".

## STEP 3

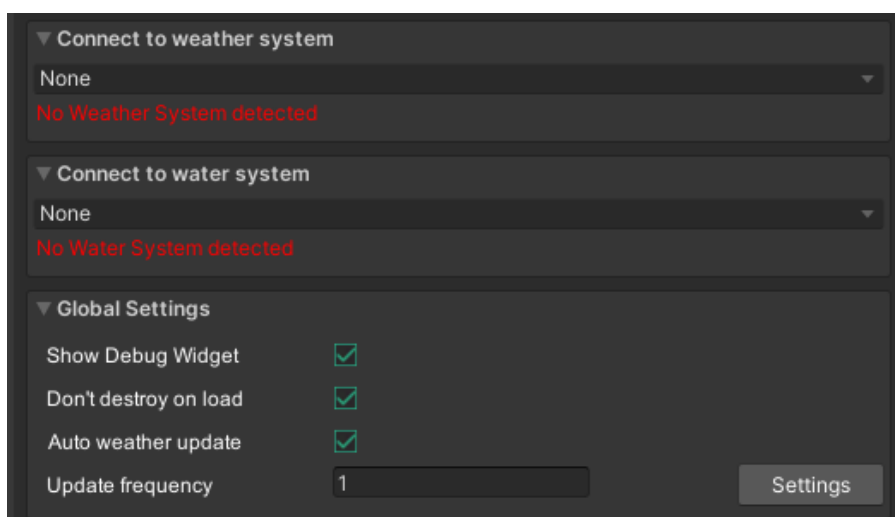
---

If you don't want the "Real-Time Weather Manager" to be destroyed when other scenes are loaded, check "Don't Destroy on Load" in the "Global Settings" panel.

## STEP 4

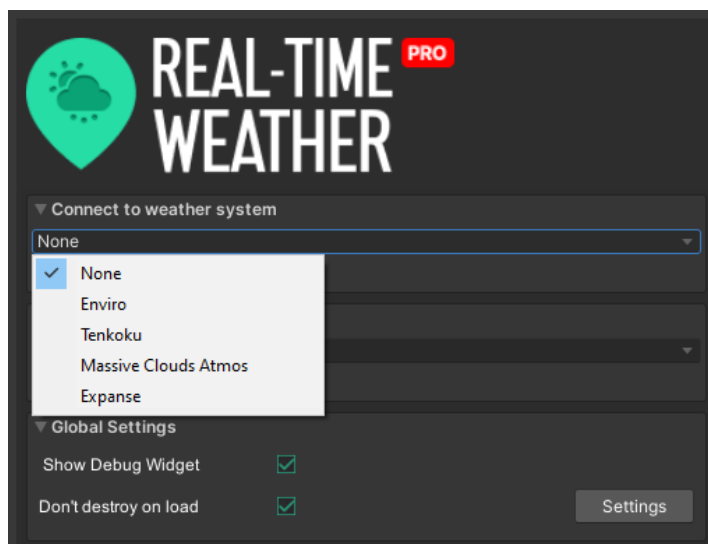
---

To get real-time weather updates, check "Auto weather update" and enter an "update frequency" that is measured in minutes. By enabling this setting, Real-Time Weather will request data from your selected providers with the given frequency.



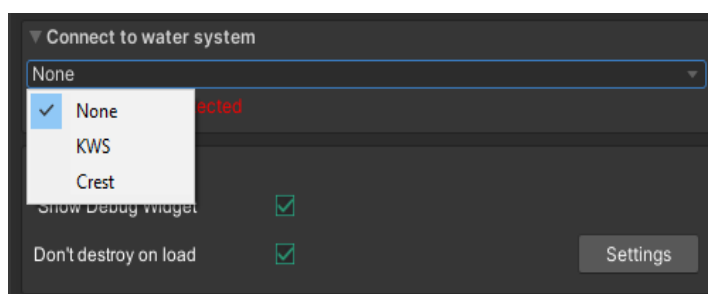
## STEP 5

Depending on what weather plugin you have in the project, you can activate either the Enviro, Tenkoku, Atmos or Expanse simulation. Simply select one from the dropdown, and everything that the weather simulation needs will be added to your scene.



## STEP 6

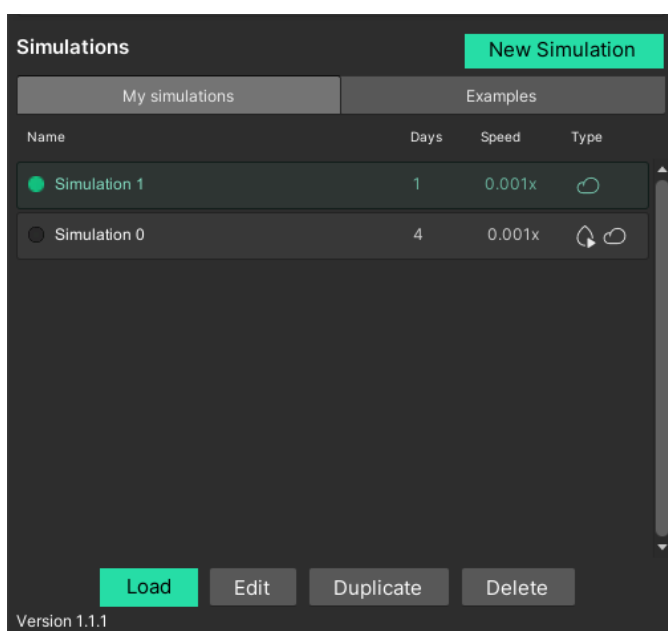
You can select from the two water systems: Crest Ocean or KWS. To activate them, simply choose one from the dropdown and everything will be added to the scene.



## STEP 7

Simulations are structures that save weather data. You can switch between your simulations and the demo simulations just by selecting one of the tabs.

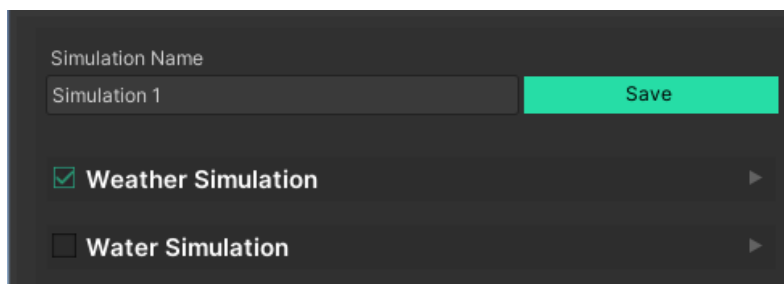
To start a simulation, just select the desired one from the list and press the load button.



## STEP 8

To create your simulation, hit the "New Simulation" button, and a new simulation will appear in the list. After that, hit the "Edit" button to customize the simulation.

In the simulation tab, you can select if you want to have the weather or water data activated, so check the data type you need.

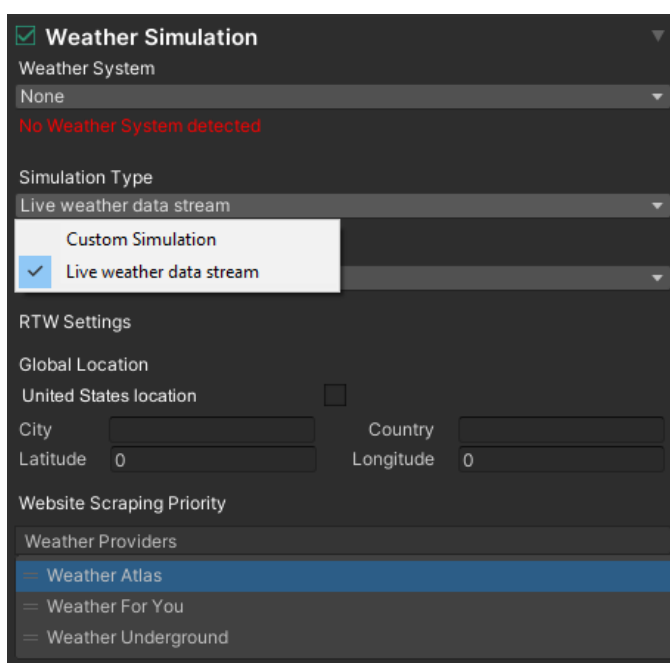


Simulation Name  
Simulation 1 Save

☒ Weather Simulation ▶

☐ Water Simulation ▶

In the weather simulation tab, select which type of data to use. You can choose from "Live weather" which is data from providers, or "Custom Simulation" where you will provide the weather data.



☒ Weather Simulation ▼

Weather System  
None ▼

No Weather System detected

Simulation Type  
Live weather data stream ▼

Custom Simulation

☒ Live weather data stream ▼

RTW Settings

Global Location

United States location ☐

City  Country

Latitude  Longitude

Website Scraping Priority

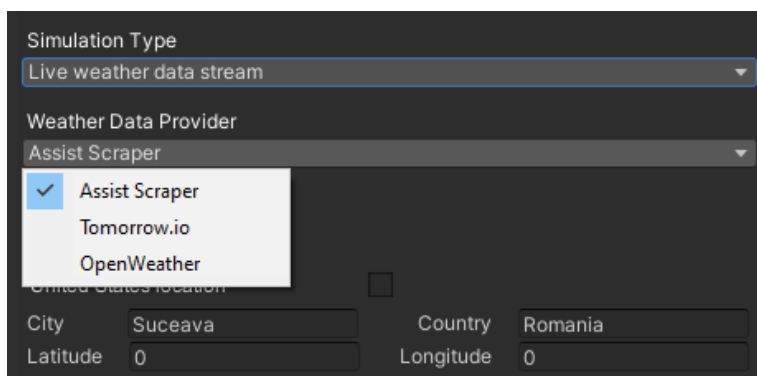
Weather Providers

☒ Weather Atlas

☐ Weather For You

☐ Weather Underground

For the live weather data, there are 3 types of providers: a data scraper, and 2 providers for which you will need an API key.



Simulation Type  
Live weather data stream

Weather Data Provider  
Assist Scraper

- ✓ Assist Scraper
- Tomorrow.io
- OpenWeather

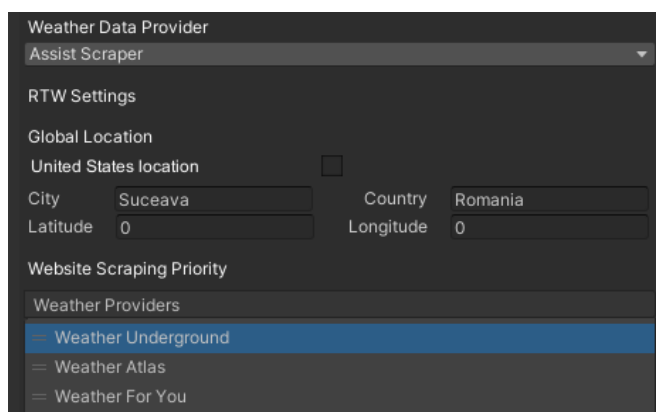
United States location: ☐

City: Suceava Country: Romania

Latitude: 0 Longitude: 0

Assist Scraper mode has 3 different weather data providers. The user can choose their priority by ordering them in the list. The first one from the top has the highest priority.

Fill in the "City" and "Country" input fields from the "Location" section to get weather data from your requested location or use geographical coordinates and type in your desired position on the globe using Latitude and Longitude information. For the USA, you need to check the "United States location" box and you will be able to fill in the "City" and "State" input fields.



Weather Data Provider  
Assist Scraper

RTW Settings

Global Location

United States location: ☐

City: Suceava Country: Romania

Latitude: 0 Longitude: 0

Website Scraping Priority

Weather Providers

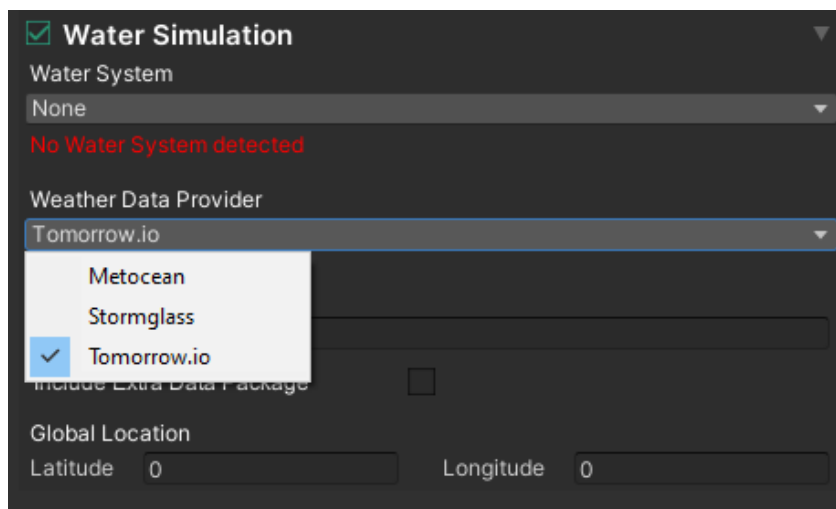
- Weather Underground
- Weather Atlas
- Weather For You



## Water Simulation

Here you can select again one of the two water systems (Crest Ocean and, KWS)

For live water data, you have 3 provider options: Metocean, Stormglass, and Tomorrow.io. All of them use an API key.



☒ **Water Simulation**

Water System  
None

No Water System detected

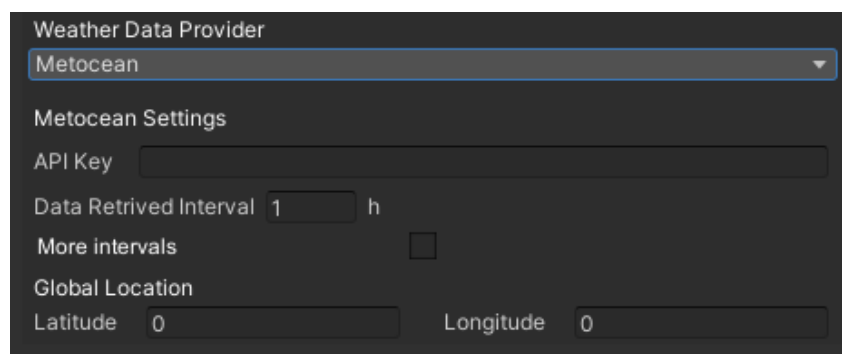
Weather Data Provider  
Tomorrow.io

- Metocean
- Stormglass
- ☒ Tomorrow.io

Include Extra Data Package ☐

Global Location  
Latitude 0 Longitude 0

Fill in the "Latitude" and "Longitude" input fields from the "Position Options" section to get maritime data from your requested geographical coordinates.



Weather Data Provider  
Metocean

Metocean Settings

API Key

Data Retrived Interval 1 h

More intervals ☐

Global Location  
Latitude 0 Longitude 0

# ENVIRO – Sky & Weather



## About

Enviro – Sky and Weather is a complete and dynamic AAA sky and weather solution! With Enviro, you can simulate weather, the day-night cycle, clouds, vegetation growth, and seasons. It's easy to set up with wonderful results. (© Hendrik Haupt).

The current Real-Time Weather version supports integration with Enviro, which can be imported from the Unity Asset Store.

### ACTIVATION

—

Real-Time Weather will automatically detect the presence of the Enviro asset and will enable the simulation controls.

Click the "Activate Enviro Simulation" button to simulate the weather using Enviro. After that, you will see that the EnviroModule object has been added to the scene. Disable any light sources.

### DEACTIVATION

—

Click the "Deactivate Enviro Simulation" button to disable weather simulation using Enviro. The EnviroModule will be deleted from the scene and the plugin will no longer receive weather data.

Now you can remove the Enviro-related objects or, if you wish, keep them for future simulations.

# Tenkoku – Dynamic Sky

## TENKOKU DYNAMIC SKY SYSTEM

### About

Tenkoku - Dynamic Sky brings completely dynamic high-fidelity sky and weather rendering to Unity developers. With Tenkoku, you can simulate weather, the day-night cycle, and clouds. It's easy to set up and it comes with wonderful results. (© Tanuki Digital).

The current Real-Time Weather version supports integration with Tenkoku, which can be imported from the Unity Asset Store.

### ACTIVATION

---

Real-Time Weather will automatically detect the presence of the Tenkoku asset and will enable the simulation controls.

Click the "Activate Tenkoku Simulation" button to simulate the weather using Tenkoku. After that, you will see that the Tenkoku Module object has been added to the scene. Disable any other light sources.

### DEACTIVATION

---

Click the "Deactivate Tenkoku Simulation" button to disable weather simulation using Tenkoku. The Tenkoku Module object will be deleted from the scene.

Now you can remove the Tenkoku asset if you wish, or keep it for future simulations.

# Massive Clouds Atmos



## About

Massive Clouds Atmos is an asset that provides the ability to render the entire sky with volumetric effects. It allows you to design the entire sky while adjusting for various weather conditions and time zones in real time. (© mewlist).

The current Real-Time Weather version supports integration with Atmos, which can be imported from the Unity Asset Store.

### ACTIVATION

—

Real-Time Weather will automatically detect the presence of the Atmos asset and will enable the simulation controls. Click the "Activate Atmos Simulation" button to simulate the weather using Atmos.

After that, you need to complete the Atmos Setup and Scene Setup as described in the Atmos Settings section below.

### DEACTIVATION

—

Click the "Deactivate Atmos Simulation" button to disable weather simulation using Atmos. The AtmosModule will be deleted from the scene and the plugin will no longer receive weather data.

The "Massive Clouds Camera Effect" script will be deleted from the camera.

## ATMOS Settings

In order to use Massive Clouds Atmos for weather simulation, the following settings must be configured:

### ATMOS SETUP WIZARD

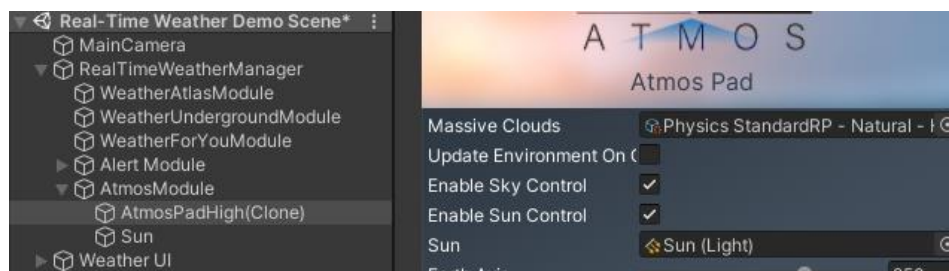
Launch the Setup Wizard from the menu: Window -> Massive Clouds Atmos -> Setup Wizard. From the Project Settings, press the "Fix Now" button on the Preloaded Shaders.

Open the Scene Setup and press the "Fix Now" button. The setup wizard supports the appropriate setup for each pipeline. Select the Renderer accordingly. For **Standalone**, select Physics StandardRP - Natural - High, and for the **Android** platform, select Physics StandardRP - Natural - Middle. The Scene setup is automatically made on Built-in pipeline.

For the Android platform, some optimizations can be done, which are described in the Atmos plugin's documentation, available here: [http://massive-clouds-atmos.mewli.st/mca\\_optimization\\_en.html](http://massive-clouds-atmos.mewli.st/mca_optimization_en.html)

### SCENE SETUP

In the scene view, select the AtmosPad object and add the reference to the light source.



Select the Camera object and check in the Inspector if it has the MassiveCloudsCameraEffect script attached. Also, check that the references to Massive Clouds and Sun are set properly.

# Expanse Integration

## EXPANSE

### About

Expanse is a state-of-the-art volumetric tool for HDRP that gives you the power to author beautiful skies, clouds, and fog banks. (© Three M's Creative).

The current Real-Time Weather version supports integration with Expanse, which can be imported from the Unity Asset Store.

### ACTIVATION

---

Real-Time Weather will automatically detect the presence of the Expanse asset and will enable the simulation controls. Click the "Activate Expanse Simulation" button to simulate the weather using Expanse.

Expanse needs to have the following modules to be detected: DateTimeController, ProceduralCloudVolume, CreativeFog, and Creative Sun. Then you will see that an Expanse Module object has been added to the scene. Disable any other light sources.

### DEACTIVATION

---

Click the "Deactivate Expanse Simulation" button to disable weather simulation using Expanse.

The Expanse Module object will be deleted from the scene. Now you can remove the Expanse asset if you wish, or keep it for future simulations.

# Crest Ocean System



## About

*Crest* is a technically advanced ocean system for Unity. [\[Link\]](#)  
It is architected for performance and makes heavy use of Level Of Detail (LOD) strategies and GPU acceleration for fast update and rendering. It is also highly flexible and allows any custom input to the water shape/foam/dynamic waves, and has an intuitive and easy-to-use shape authoring interface. (© Wave Harmonic)

The current Real-Time Weather version supports integration with Crest version 4.15.2, which can be imported from the Unity Asset Store. [\[Link\]](#)

## ACTIVATION

---

Real-Time Weather will automatically detect the presence of the Crest asset in the Water Simulation tab and will enable the simulation controls.

Click the "Activate Crest Simulation" button to simulate maritime data using Crest. After that, the CrestModule object will be added to the scene. While using High Definition RP, if there is no light source in the scene, Crest will log an error asking for a primary light source inside its OceanRender component.

## DEACTIVATION

---

Click the "Deactivate Crest Simulation" button to disable Crest Ocean simulation. The CrestModule will be deleted from the scene and the plugin will no longer receive maritime and weather data.



# KWS Integration

## About

KWS allows you to simulate water surfaces such as those of ocean, sea, rivers, lakes, pools, etc. (© krypto289)

## ACTIVATION

---

Real-Time Weather will automatically detect the presence of the [KWS Water System 1.3.0 \(URP\)](#) or [KWS Water System 1.3.0 \(HDRP\)](#) assets and will enable the respective simulation controls. Click the “Activate KWS Simulation” button to simulate the water using KWS.

Once the water system has been activated, it will search for all the compatible present water systems in the scene and list them in the KWSModule gameobject that has been created as a child of the RealTimeWeatherManager. You will need to specify which type of water body each water system is for the simulation to work properly.

## DEACTIVATION

---

Click the “Deactivate KWS Simulation” button to disable water simulation using KWS.

The KWSModule object will be deleted from the scene. Now you can remove the KWS asset if you wish or keep it for future simulations.



# Forecast Timelapse

## About

"Forecast Timelapse" is a feature that allows the user to create a simulation of the atmospheric conditions for several hours into the future. This works by fetching weather data from one of the supported providers and creating a speed-up simulation based on that data.

## ACTIVATION

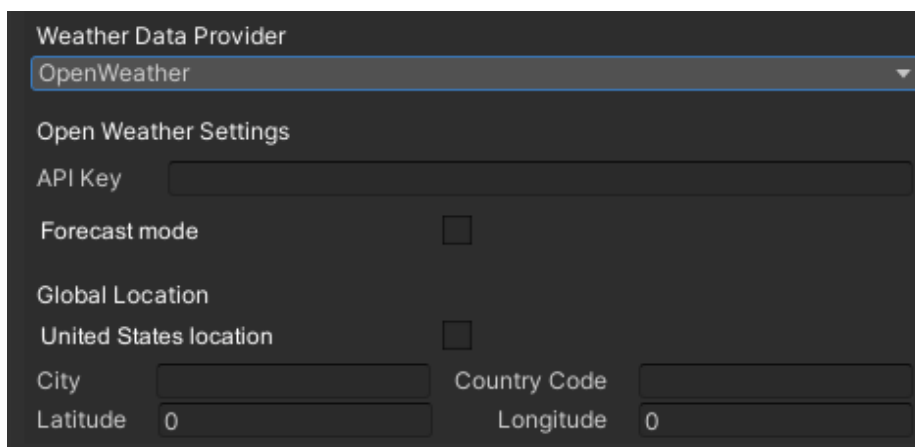
—

Currently, you can use this feature for user-data forecast simulations, OpenWeatherMap, and Tomorrow.io. We will go into detail on how to set up the forecast mode for each of these providers.

### Working with OpenWeatherMap

To set up forecasting mode for OpenWeatherMap, first we need to select the provider from the "Weather Data Provider" dropdown list in the "Live Weather Data stream" section.

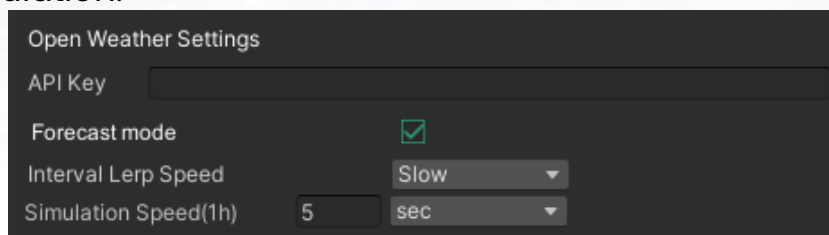
After setting up the "API Key" parameter, we need to input the latitude and longitude values of the location that we will use for our weather simulation.



The screenshot shows a configuration panel for OpenWeatherMap. At the top, a dropdown menu labeled "Weather Data Provider" has "OpenWeather" selected. Below this, the "Open Weather Settings" section contains an "API Key" text input field. Underneath, there is a "Forecast mode" checkbox which is currently unchecked. The "Global Location" section includes a "United States location" checkbox, also unchecked. At the bottom, there are four input fields: "City", "Country Code", "Latitude" (with the value "0"), and "Longitude" (with the value "0").

# Forecast Timelapse

After setting up these options, we need to activate the "Forecast Mode" in the "Forecast Weather Settings" section at the bottom. Enabling forecast mode will bring up these controls that allow the user to adjust the forecast simulation.

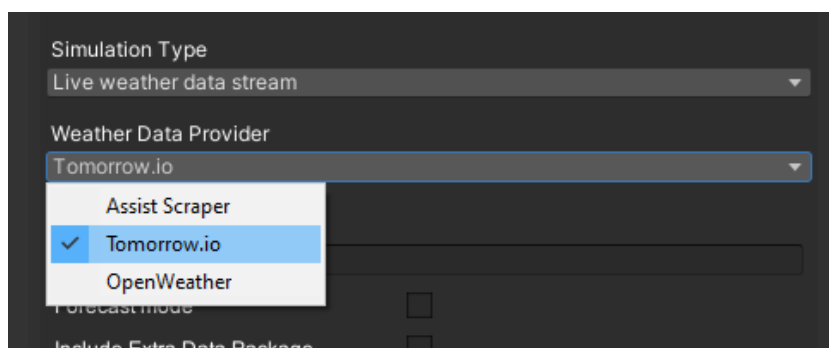


From here we can set a few simulation parameters:

- Forecast Simulation Speed: a value in seconds that corresponds to one (1) hour in real-world time;
- Lerp Speed: options -> "Slow", "Medium", and "Fast", the amount of linear interpolation applied between two intervals of time (hours or days) so that we can have smooth transitions between weather conditions.
- Loop: the obtained forecast can be simulated in a continuous cycle, where the forecast data is reused at every step, but the time continues to advance with the passing of every simulation loop.

## Working with Tomorrow.io

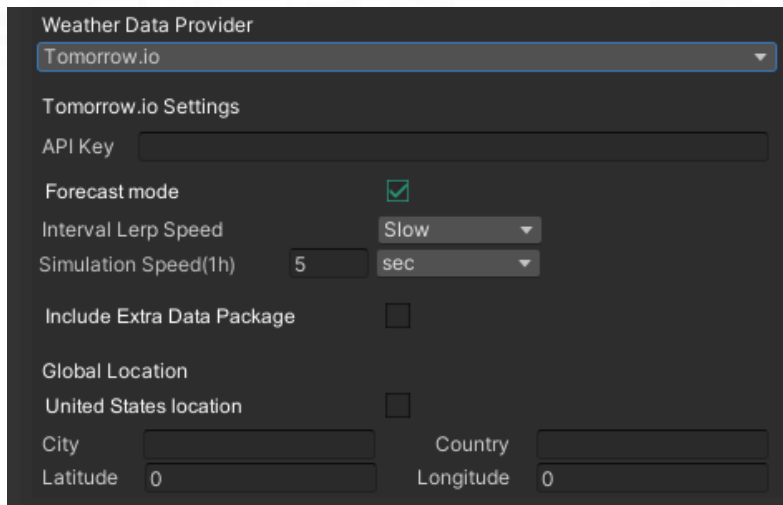
In order to set up the forecasting mode for Tomorrow.io, first we need to select the provider from the "Request Mode" dropdown list in the "General Settings" section.



# Forecast Timelapse

After setting up the "API Key" parameter, we need to input the location that we will use for our weather simulation.

Lastly, we can enable the "Forecast Mode" option, which will toggle a set of simulation parameters that we can adjust.

The image shows a dark-themed settings window for a weather simulation. At the top, there's a dropdown menu labeled "Weather Data Provider" with "Tomorrow.io" selected. Below this is a section titled "Tomorrow.io Settings". It contains an "API Key" text input field. Underneath is a "Forecast mode" section with a checked checkbox. Below that are two dropdown menus: "Interval Lerp Speed" set to "Slow" and "Simulation Speed(1h)" set to "5" with a unit dropdown set to "sec". There's an "Include Extra Data Package" checkbox which is unchecked. A "Global Location" section follows, with an unchecked "United States location" checkbox. At the bottom, there are input fields for "City", "Country", "Latitude" (set to "0"), and "Longitude" (set to "0").

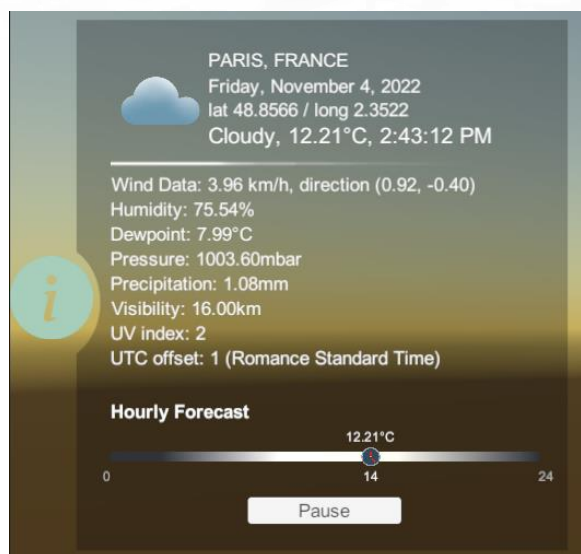
From here, we can set a few simulation parameters:

- Forecast Simulation Speed: a value in seconds that corresponds to one (1) hour in real-world time;
- Lerp Speed: options -> "Slow", "Medium", and "Fast"; the amount of linear interpolation applied between two intervals of time (hours or days), so that we can have smooth transitions between weather conditions.
- Loop: The obtained forecast can be simulated in a continuous cycle, where the forecast data is reused at every step, but the time continues to advance with the passing of every simulation loop.

# Forecast Timelapse

## Starting the forecast timelapse

After going through the setup of one of the providers, we can start the simulation by hitting "Play" in the Unity Editor. The UI info panel will change its layout to display either a 24h timeline or a 7-day forecast based on the forecast mode selected earlier.



Hourly Forecast layout



Daily Forecast layout

As the simulation time progresses, the forecast timeline and table will be updated to highlight the current time increment (hour or day).

During the forecast simulation, the user has the option to stop or resume the forecast using the available "Pause" or "Resume" buttons on the UI info panel.

You can listen to the Forecast Module events to obtain forecast updates:

```
public static event Action<WeatherData, WeatherData, double> OnForecastProgressModuleTick;
```

### Example:

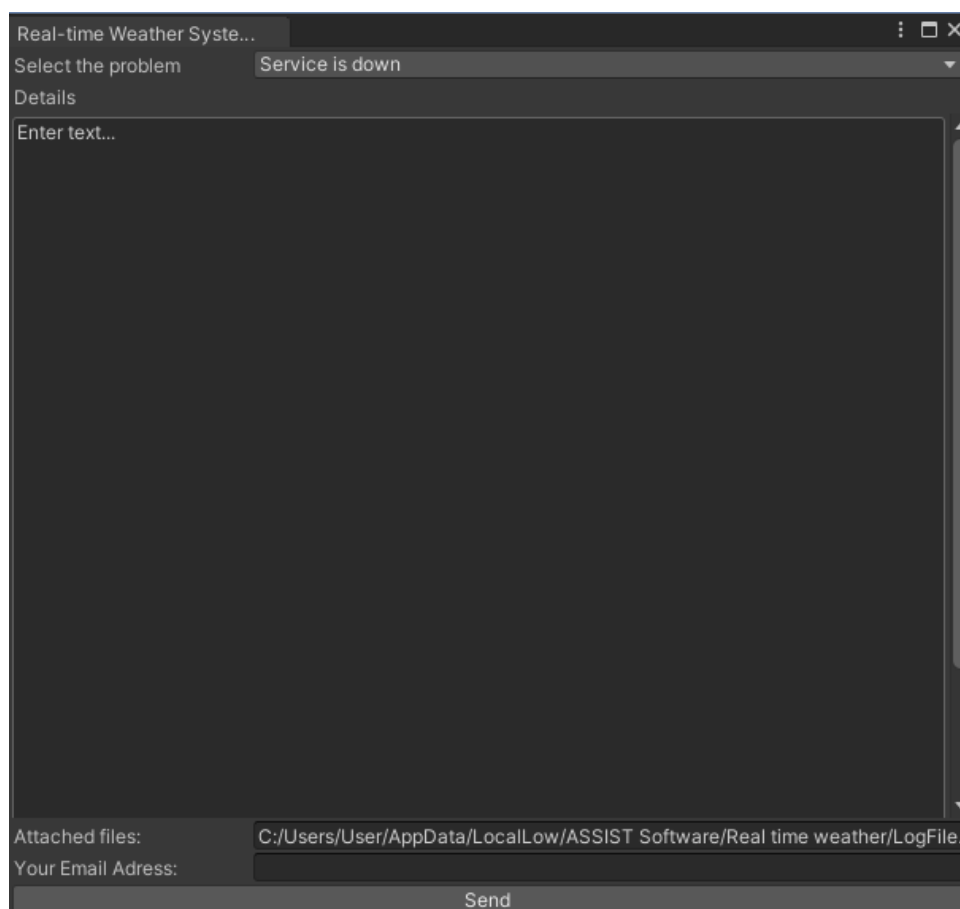
```
ForecastModule.instance.OnForecastProgressModuleTick += OnForecastUpdate;
```

# Help & Bug Report

## Report

This module will send an email to the Real-Time Weather developers when an error appears in plugins used for getting weather data or when a user wants to send an email with some questions.

This pop-up is automatically activated when all the weather modules fail to obtain weather data. It can also be activated by going to the “Real-time Weather Manager” tab and selecting “Help”, then “Bug Report”.



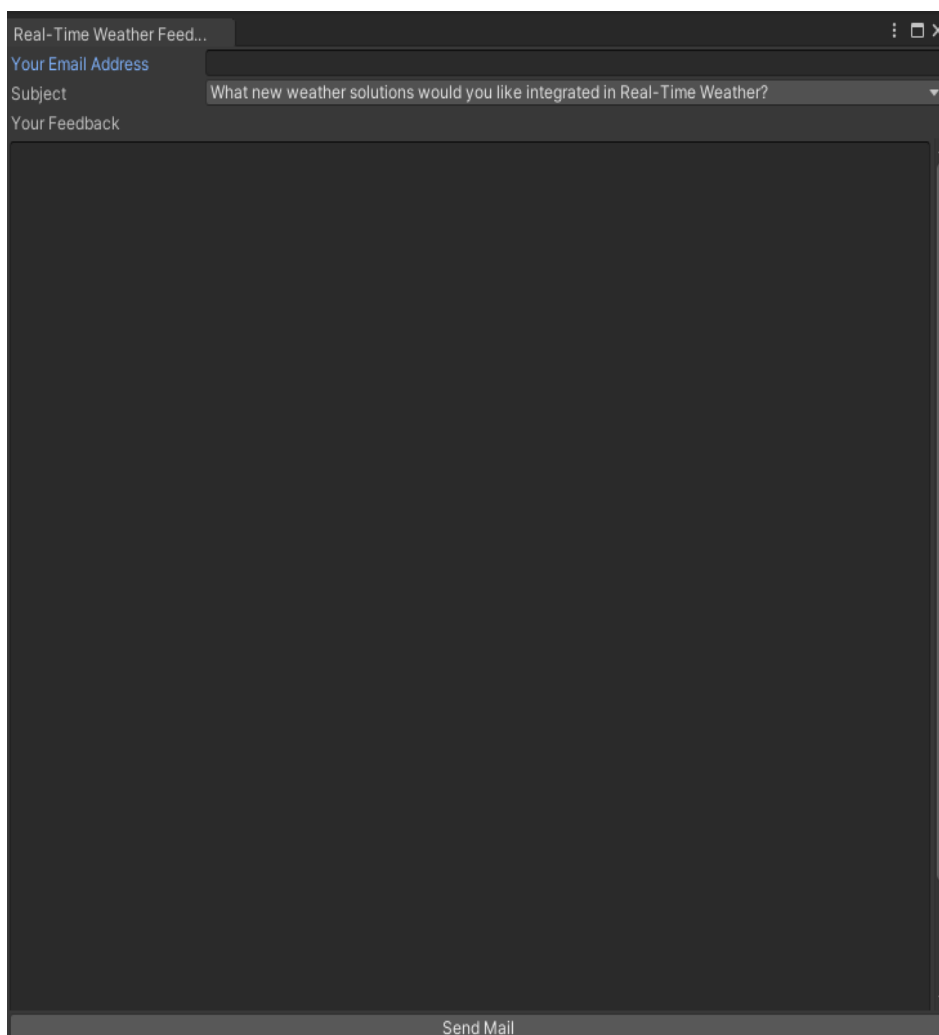
The screenshot shows a dark-themed dialog box titled "Real-time Weather System...". It features a "Select the problem" dropdown menu with "Service is down" selected. Below this is a "Details" section with a large text area labeled "Enter text...". At the bottom, there are fields for "Attached files:" (showing a file path) and "Your Email Address:", followed by a "Send" button.

# Feedback Form

## Feedback

This dialog can be opened from the toolbar by going to the “Real-Time Weather” tab and selecting “Help”, then “Feedback Form”.

The module is used to send your feedback to the Real-Time Weather developers.



The screenshot shows a dark-themed dialog box titled "Real-Time Weather Feed...". It contains three main input areas: "Your Email Address" (a text field), "Subject" (a dropdown menu with the text "What new weather solutions would you like integrated in Real-Time Weather?"), and "Your Feedback" (a large text area). At the bottom right, there is a "Send Mail" button.

## FAQs

**Q: I don't know how to use the Real-Time Weather PRO plugin. Do you have a tutorial?**

A: Yes, you can check out our tutorials from Unity Asset Store product page. You can also read the **Overview & Installation** chapter from the documentation, which has all the information you need for any setup.

**Q: Which pipeline renderers are supported?**

A: Real-Time Weather Manager works on any rendering pipeline, but you should be careful about the weather systems or water systems you use. For example, the Tenkoku asset works only on SRP, and the Expanse asset works only on HDRP.

**Q: Does this asset provide forecast simulations?**

A: Yes, you can simulate complete forecast scenarios (from weather providers, or forecast presets created by you), but only for weather data.

**Q: I activated Massive Clouds Atmos and no weather is simulated.**

A: Ensure that you configured all the necessary settings for Atmos (sun settings, pipeline settings). You can find them in this documentation, at Massive Clouds Atmos integration section.

**Q: I'm using the Real-Time Weather mode, but I'm not receiving any weather data.**

A: If the weather data for some specific locations is not returned from weather providers, then change the request to a nearby location. Also, we are happy to be contacted through the **Real-Time Weather/Help/Bug Report** window, to investigate any issues you encounter.

**Q: Can I use the requested data for my implementations?**

A: Of course, you can subscribe to the event that returns the weather data and use the data in any way you want.

**Q: How frequently can the weather data be updated?**

A: For the current weather mode, you can update the weather data at a 1-minute rate, while the maximum time between two updates can be up to 2 hours.

**Q: Is there any limit on the number of requests I can make ?**

A: You have 3 providers for weather data. The first one (Real-Time Weather mode), that has no limits, and the API providers (Tomorrow.io and OpenWeatherMap) have different limits based on the subscription. You can choose to request maritime data from Metocean, Tomorrow.io or Stormglass APIs, which also have some requests limits based on the membership.

**Q: What can I do if I discover a bug?**

A: We will be pleased to be contacted through the "Bug Report" window, to resolve the issue. You can access the window from the Unity Menu Bar: **Real-Time Weather/Help/Bug Report**.

**Q: When I use the Tenkoku asset to simulate weather, there's a black line along the horizon, visible during the night-time.**

A: The visual artifact can be observed due to the Tenkoku clouds, which extend to the high distances, and is more visible when the ground is positioned above the pivot of the clouds object. The developer team suggests that in this case, you can lower the clouds position (Y-Position value) from Tenkoku -> SkySphere -> Effects -> fxCloudSphere gameobject.



# Scripting API

## Real-Time Weather Manager

The `RealTimeWeatherManager` class is implemented using the Singleton design pattern and it manages the main functionalities of the Real-Time Weather plugin.

It allows the weather data requests from the Atlas , Underground , and Weather For You modules, and it also manages the automatic weather data update and weather simulation using third-party support components: Enviro, Tenkoku, Massive Clouds Atmos, and Expanse.

### REQUEST WEATHER DATA

---

The weather data request can be made using the following function:

```
public void RequestWeatherByCityAndCountry(string city, string country)
```

**Example:**

```
RealTimeWeatherManager.instance.RequestWeatherByCityAndCountry("Paris", "France");
```

### RECEIVE WEATHER DATA

---

Receiving current weather data can be done by subscribing to the `OnCurrentWeatherUpdate` event. To receive weather forecast data (hourly or daily), a subscription to `OnHourlyWeatherUpdate` and/or `OnDailyWeatherUpdate` is required.

```
public delegate void CurrentWeatherUpdate(WeatherData weatherData);  
public delegate void HourlyWeatherUpdate(List<WeatherData> weatherData);  
public event CurrentWeatherUpdate OnCurrentWeatherUpdate;  
public event HourlyWeatherUpdate OnHourlyWeatherUpdate;
```

**Example:**

```
RealTimeWeatherManager.instance.OnCurrentWeatherUpdate +=  
OnCurrentWeatherUpdate;  
RealTimeWeatherManager.instance.OnHourlyWeatherUpdate +=  
OnHourlyWeatherUpdate;
```

## NOTIFY WEATHER DATA CHANGED

---

Send notifications with updated weather data to the components that listen to the `OnCurrentWeatherUpdate`, `OnHourlyWeatherUpdate`, and `OnDailyWeatherUpdate` events.

```
private void NotifyCurrentWeatherChanged(WeatherData weatherData);  
private void NotifyHourlyWeatherChanged(List<WeatherData> weatherData);  
private void NotifyDailyWeatherChanged(List<WeatherData> weatherData);
```

## Weather data

The weather data class is used to store and manage weather data.

- *Localization* is a `Localization` class instance that holds the localization data: city, country, latitude, and longitude.
- *DateTime* is an instance of the `DateTime` structure that represents an instant in time, typically expressed as a date and time of day.
- *UtcOffset* is an instance of `TimeSpan` structure that represents the difference in hours and minutes from Coordinated Universal Time (UTC) for a particular place and date.



- *Wind* is a Wind class instance that holds the wind data: direction and speed. Speed is measured in km/h.
- *Temperature* is a float value that represents the temperature in °C.
- *WeatherState* is a WeatherState enum value that represents the weather state.

```
public enum WeatherState
{
    Clear,
    PartlyClear,
    Cloudy,
    PartlyCloudy,
    Mist,
    Thunderstorms,
    RainSnowPrecipitation,
    RainPrecipitation,
    SnowPrecipitation,
    Windy,
    PartlySunny,
    Sunny,
    Fair
}
```

- *Pressure(atmospheric pressure)*, also known as barometric pressure, is a float value that represents the pressure within the atmosphere of Earth measured in millibars.
- *Humidity* is a float value that represents the humidity as a percentage.
- *TimeZone* represents the current time zone of the searched location in the following format: "Continent/Country".
- *Precipitation* is a float value that represents the precipitation in mm.
- *Dewpoint* is the temperature in °C to which air must be cooled to become saturated with water vapor.
- *Visibility* is a float value that represents the visibility in km.

- *IndexUV* is a float value that represents the UV(ultraviolet) index. The ultraviolet index is an international standard measurement of the strength of sunburn-producing ultraviolet radiation at a particular place and time.

The weather data received from the services can be viewed in the WeatherUI interface.



The WeatherUI Prefab can be found in Real-Time Weather Manager/Prefabs/UI Prefabs. The information will be displayed if the simulation settings have been set.

## ATLAS module

This module is responsible for downloading web pages from <https://www.weather-atlas.com> and parsing them to obtain weather information.

The webpage data is requested using an input containing the non-abbreviated, complete names of the city and country, for example: "Spain/Madrid".

The obtained data is parsed through the HtmlAgilityPack HTML parser, a C# linked solution that transforms a plain text into a parsable HTML node structure using tags division.

### RECEIVE ATLAS WEATHER DATA

Receiving Atlas weather data can be done by subscribing to the OnWebPageParsed delegate.

```
public delegate void OnWebPageParsed(WeatherData webPageData);  
public onWebPageParsed;
```

**Example:**

```
_atlasModule.onWebPageParsed += OnReceivingAtlasWeatherData;
```

### RECEIVE ATLAS EXCEPTIONS

Receiving Atlas exceptions can be done by subscribing to the OnExceptionRaised delegate.

```
public delegate void OnExceptionRaised(ExceptionType type, string message);  
public onExceptionRaised;
```

**Example:**

```
_atlasModule.onExceptionRaised += OnRequestWeatherServiceExceptionRaised;
```

The WeatherAtlasModule is instantiated in the scene as a child of RealTimeWeatherManager. You can use this module separately. The WeatherAtlasModulePrefab prefab can be found in Real-Time Weather/Prefabs/Weather Modules Prefabs.

## Weather for You module

This module is responsible for downloading web pages from <https://www.weatherforyou.com/> and parsing them to obtain weather information.

The webpage data is requested using input data composed of the name of the city/county and the abbreviated name of the country, for example: *"&place=liverpool&state=&country=gb"*.

The obtained data is parsed through the HtmlAgilityPack HTML parser, a C# linked solution that transforms a plain text into a parsable HTML node structure using tags division.

### RECEIVE WEATHERFORYOU WEATHER DATA

—

Receiving WeatherForYou weather data can be done by subscribing to the OnWebPageParsed delegate.

```
public delegate void OnWebPageParsed(WeatherData webPageData);  
public onWebPageParsed;
```

**Example:**

```
_weatherForYou.onWebPageParsed += OnReceivingWeatherForYouData;
```

## RECEIVE WEATHERFORYOU EXCEPTIONS

---

Receiving WeatherForYou exceptions can be done by subscribing to the OnExceptionRaised delegate.

```
public delegate void OnExceptionRaised(ExceptionType type, string message);  
public onExceptionRaised;
```

**Example:**

```
_weatherForYou.onExceptionRaised += OnRequestWeatherServiceExceptionRaised;
```

The WeatherForYouModule is instantiated in the scene as a child of RealTimeWeatherManager. You can use this module separately. The prefab can be found in Real-Time Weather/Prefabs/Weather Modules Prefabs.

## Weather Underground module

This module is responsible for downloading web pages from <https://www.wunderground.com/> and parsing them to obtain weather information.

The webpage data is requested using input data composed of the abbreviated name of the country and the name of the city/county, for example: "fr/Paris".

The obtained data is parsed through the HtmlAgilityPack HTML parser, a C# linked solution that transforms a plain text into a parsable HTML node structure using tags division.

## RECEIVE WEATHER UNDERGROUND WEATHER DATA

---

Receiving Weather Underground weather data can be done by subscribing to the `OnWebPageParsed` delegate.

```
public delegate void OnWebPageParsed(WeatherData webPageData);  
public onWebPageParsed;
```

**Example:**

```
_undergroundModule.onWebPageParsed += OnReceivingUndergroundData;
```

## RECEIVE WEATHER UNDERGROUND EXCEPTIONS

---

Receiving Weather Underground exceptions can be done by subscribing to the `OnExceptionRaised` delegate.

```
public delegate void OnExceptionRaised(ExceptionType, string message);  
public onExceptionRaised;
```

**Example:**

```
_undergroundModule.onExceptionRaised += OnRequestWeatherServiceExceptionRaised;
```

The `WeatherUndergroundModule` is instantiated in the scene as a child of `RealTimeWeatherManager`. You can use this module separately.

The `WeatherUndergroundModulePrefab` prefab can be found in `Real-Time Weather/Prefabs/Weather Modules Prefabs`.



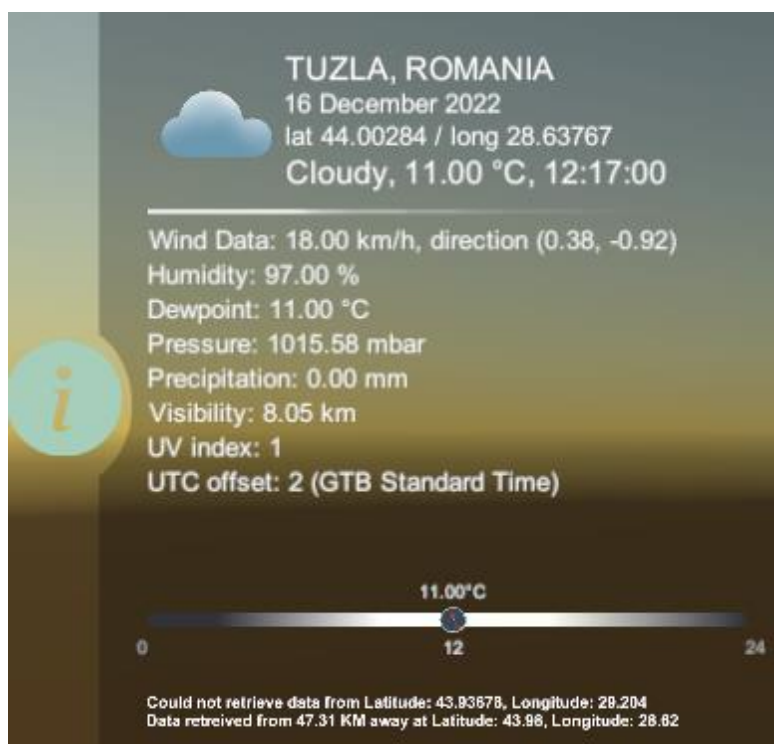
## Reserve Locations (only for Geographic Coordinates input)

When the user-given coordinates can't be geo-located, a new location will be provided, near to the given coordinates.

The new location will be chosen from a list of 9000 known weather stations around the world. The list has been generated from Greg Thompson's file [\[WeatherStations\]](#).

Up to five of the closest weather stations will be provided as replacements for the user-given coordinates. If all the replacement request attempts fail, an error will appear in the RTW UI.

If one of the five weather stations returns a valid geo-location, the user will be warned that the initial given coordinates couldn't be located and the reserve location has been chosen, displaying also the distance between them.



# Open Weather Map Service



## About

OpenWeatherMap [[Link](#)] is a paid service that offers the opportunity to obtain real-time weather data through miscellaneous HTTP requests to its server. The server can be requested through various [[APIs](#)] using the API key specific to your account. Real-Time Weather Manager uses the [[Current Weather Data API](#)] and [[One Call API](#)].

The request is composed of: API key, localization data, and optional parameters.

## Current weather

```
api.openweathermap.org/data/2.5/weather?q={city  
name}&appid={API key}
```

## REQUEST METHODS

---

There are 4 modes of request:

- City name, state code (applicable only for the US), and country code
- City ID [cities.json]
- Latitude & longitude
- Zip code and country code

## PARAMETERS

---

There are 2 types of parameters:

- Language: en, fr, de, jp, ro, etc.
- Request mode represents the 4 types of requests presented.

## Weather forecast for a period of time

For Geographic Coordinates Request mode, the weather forecast data can be requested for the next 48 hours with a step of one hour.

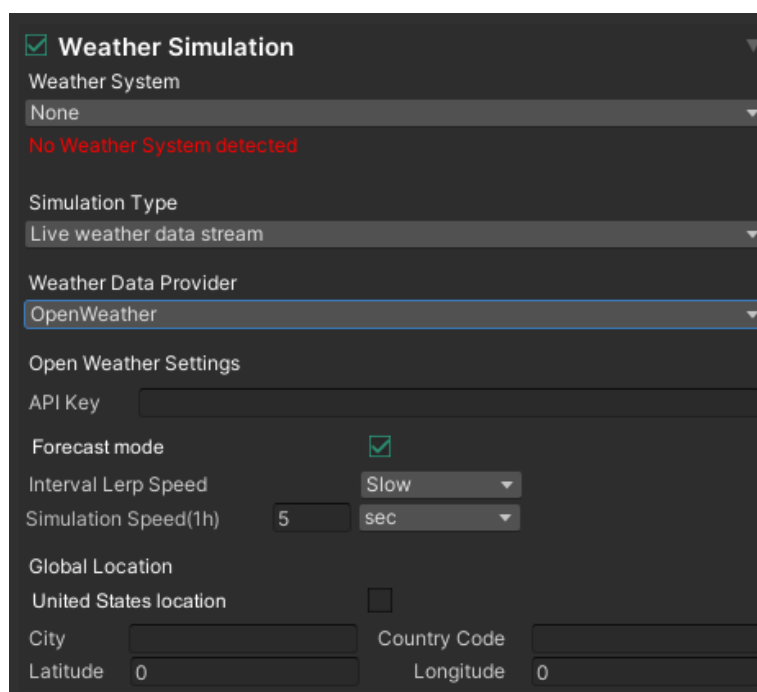
The following link is used:

```
https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude={part}&appid={API key}
```

## Parameters

There is an additional optional parameter that can exclude some parts of the weather data from the API response. The parameter is **exclude**, and its value is a comma-delimited list. Available values are current, minutely, hourly, daily, and alerts. It has 3 default values: **current**, **minutely**, and **alerts**. If weather data for the next 48 hours is wanted, the exclude parameter will also contain the **daily** value.

## Inspector interface



The Inspector interface for Weather Simulation is shown. It includes a dropdown for Weather System (None), a message 'No Weather System detected', a dropdown for Simulation Type (Live weather data stream), a dropdown for Weather Data Provider (OpenWeather), and a section for Open Weather Settings. The Open Weather Settings section includes an API Key field, a Forecast mode checkbox (checked), an Interval Lerp Speed dropdown (Slow), a Simulation Speed(1h) field (5) with a unit dropdown (sec), a Global Location section with a United States location checkbox, and fields for City, Country Code, Latitude (0), and Longitude (0).

## Openweather map data

The OpenWeatherMap Data class is used to store and manage the weather data obtained by the service:

- *The Geographic Coordinates* member class holds details about the geographic positioning on the globe: latitude and longitude.
- *List<WeatherDetails>* is a list with *WeatherDetails* instances. Every instance has the following members: ID (code of the weather state), main (the weather state in string format), description (additional details) and icon (a specific icon that appears on their webpage).
- *Base* represents the source from where the data was obtained => stations, statistics.
- *Main Weather* is a class instance that holds the main weather details: temperature, minimum temperature, maximum temperature, pressure, humidity, temperature feels\_like, pressure at sea level and pressure at ground level.
- *Visibility* is a float value that represents the distance at which an object or light can be clearly discerned.
- *Wind* is a Wind class instance that holds the wind data: direction, speed and gust.
- *Clouds* is a Clouds instance that holds cloud density data (percentage) .
- *Unix timestamp (dt)* is a long value that represents a way to track time as a running total of seconds. This value is added to epoch time 1st Jan, 1970) and constructs the current date time.
- *System Data* is a class that holds some specific system parameters such as type, ID, message (logs), country (the interrogated country), sunrise time, and sunset time.
- *Timezone* represents the UTC offset in seconds. Example: for UTC +3:00 hours => 3 \* 3600 seconds => 10800 seconds.

- *CityID* is an int value that represents the corresponding city id for the interrogated location, which can be found on [city.list.json.gz](http://city.list.json.gz).
- *HTTPCode* is an int value that represents the HTTP code response from the server (200 => OK, 404 => Not Found, etc.).
- *City name* is a string value that represents the city from the interrogated location. This can be useful, for example, when we request data using latitude & longitude.
- *Units* is a Units enum value that represents the units the data will be obtained in: **Standard** (speed => meter/sec, temperature => Kelvin), **Metric** (speed => meter/sec, temperature => Celsius) or **Imperial** (speed => miles/hour, temperature => Fahrenheit).

## OpenWeather One Call API Map Data

The `OpenWeatherOneCallAPIMapData` Data class is used to store and manage the weather data obtained from the One Call API service:

- *Unix timestamp (dt)* is a long value that represents a way to track time as a running total of seconds. This value is added to epoch time (1<sup>st</sup> Jan, 1970) and constructs the current date time.
- *The Geographic Coordinates* member class holds details about the geographic positioning on the globe: latitude and longitude.
- *TimezoneOffset* represents the UTC offset in seconds. Example: for UTC +3:00 hours => 3 \* 3600 seconds => 10800 seconds.
- *List<HourlyWeather>* is a list with *HourlyWeather* instances. Every instance holds the main weather details: unix timestamp, temperature, temperature feels\_like, pressure at sea level and pressure at ground level, humidity, dew point, UV Index, clouds, visibility, wind speed, wind gust, wind degree, and a list of *WeatherDetails*.
- *Timezone* represents the name for the requested location.

- *List<DailyWeather>* is a list with *dailyWeather* instances. Every instance holds the main weather details: unix timestamp, temperature, temperature feels\_like, pressure at sea level and pressure at ground level, humidity, dew point, UV Index, clouds, wind speed, wind gust, and a list of WeatherDetails.
- *Units* is a Units enum value that represents the units the data will be obtained in: **Standard** (speed => meter/sec, temperature => Kelvin), **Metric** (speed => meter/sec, temperature => Celsius) or **Imperial** (speed => miles/hour, temperature => Fahrenheit).

## Redirecting data outside our plugin

The module parses the weather data and afterwards, sends it to the manager, but it can also be directed to other plugins or your solution.

### RECEIVE OPENWEATHERMAP DATA

Receiving OpenWeatherMap data can be done by subscribing to the OnServerResponse delegate from the OpenWeatherMapModule class.

```
public delegate void OnServerResponse(OpenWeatherData weatherData);  
public OnServerResponse onServerResponse;
```

**Example:**

```
_openWeatherMapModule.onServerResponse += OnReceivingOpenWeatherMapData;
```

## RECEIVE OPENWEATHERMAPONECALLAPI DATA

---

Receiving OpenWeatherOneCallAPIMap data can be done by subscribing to the OnServerOneCallAPIResponse delegate from the OpenWeatherMapModule class.

```
public delegate void OnServerOneCallAPIResponse (OpenWeatherOneCallAPIMapData
weatherData);
public OnServerOneCallAPIResponse onServerOneCallAPIResponse;
```

**Example:**

```
_openWeatherMapModule.onServerOneCallAPIResponse +=
    OnReceivingOpenWeatherMapOneCallAPIData;
```

## RECEIVE OPENWEATHERMAP EXCEPTIONS

---

Receiving OpenWeatherMap exceptions can be done by subscribing to the OnExceptionRaised.

```
public delegate void OnExceptionRaised(ExceptionType type, string message);
public OnExceptionRaised onExceptionRaised;
```

**Example:**

```
_openWeatherMapModule.onExceptionRaised += OnOpenWeatherMapExceptionRaised;
```

## Redirecting data outside our plugin

As long as Real-Time Weather Manager requests weather data from Open Weather Map, then a class named Open Weather Map Converter will be responsible for converting the Open Weather Data structure to our default weather data structure.

To convert Weather Forecast data for a period of 48 hours or 7 days, a class named Open Weather One Call API Map Converter will be responsible for converting the Open Weather One Call API Data structure to a list of Weather Data structures for each case (48 hours or 7 days).

# Tomorrow



## About

Tomorrow.io is the world's leading All-in-One Weather Intelligence Platform™ [\[Link\]](#), which offers the opportunity to obtain real-time weather/maritime data.

The Tomorrow API is organized in a RESTful, stable endpoint structure, administered over HTTPS response codes and authentication. The API has predictable URLs, request query and body parameters, and JSON-encoded responses.

Access to the Tomorrow API requires a valid access key with the right permissions.

## REQUEST METHODS

---

To request weather & maritime data from Tomorrow, you must configure the following settings in the Tomorrow Module inspector:

- Specify a valid *API key* in the Tomorrow API key field. Requests not properly authenticated will return a 403-error code;
- Specify *latitude* and *longitude* (ISO 6709).

## PARAMETERS

---

The Tomorrow API will automatically request the core weather data such as temperature, wind speed and direction, and so on. In the "Weather Data Settings" menu, check what forecast information and what additional information you want to request from the API. You can choose from a set of weather parameters related to forecast, air quality, and pollen.

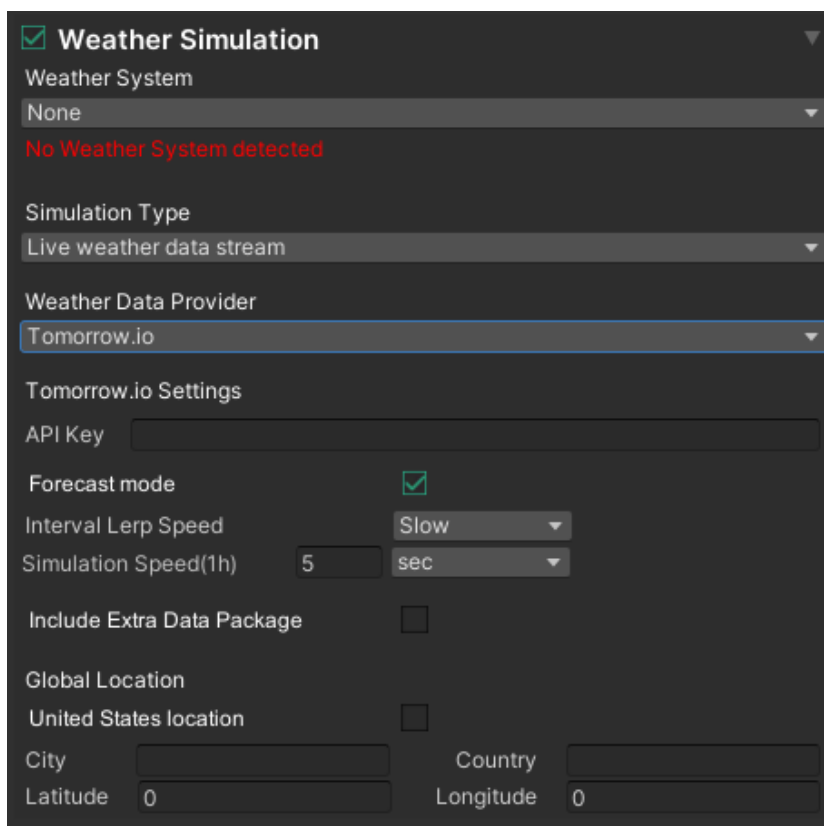
The Tomorrow API also supports maritime data for ocean and sea locations.



## Inspector interface – Weather Data

- *API Settings* – Tomorrow API settings, such as access key;
- *Location Settings* – settings for the location from which the weather information will be requested;
- *Weather Data Settings* – settings that define what weather parameters will be requested from the API.

Additional parameters that provide forecast and meteorological information related to air quality and pollen, must be checked ("Extra Data Package") to be requested from the API. Otherwise, these parameters will have default values.



☒ **Weather Simulation**

Weather System  
None

No Weather System detected

Simulation Type  
Live weather data stream

Weather Data Provider  
Tomorrow.io

Tomorrow.io Settings

API Key

Forecast mode ☒

Interval Lerp Speed Slow

Simulation Speed(1h) 5 sec

Include Extra Data Package ☐

Global Location

United States location ☐

City Country

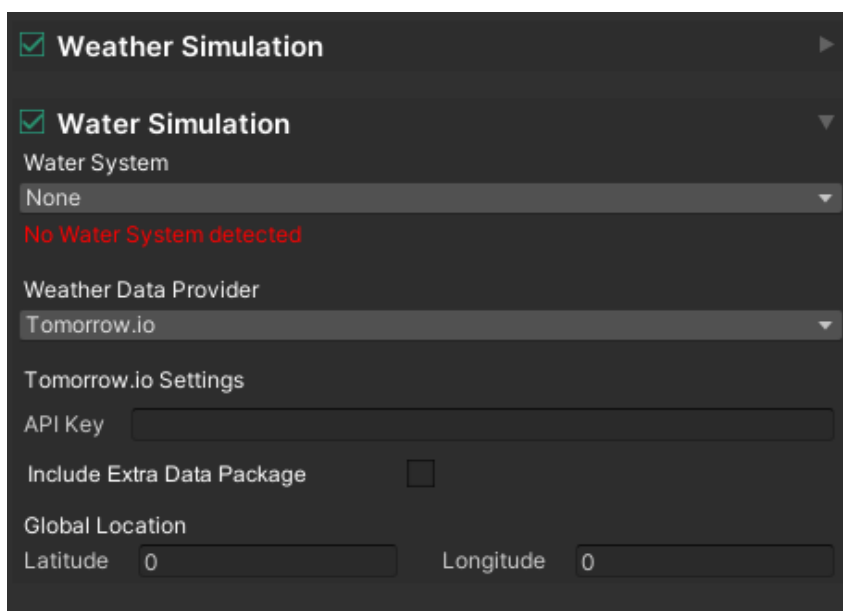
Latitude 0 Longitude 0

## Inspector interface – Maritime Data

The TomorrowWaterModule works in a similar way to the TomorrowWeatherModule, adding maritime data variables to the request.

The three main options are:

- *API Settings* – Tomorrow API settings, such as access key;
- *Location Settings* – settings for the location from which the weather and maritime information will be requested;
- *Maritime Data Settings* – additional parameters that can be requested from the API.



The image shows a dark-themed settings panel for the Inspector interface. It contains the following elements:

- ☒ **Weather Simulation** (with a right-pointing arrow)
- ☒ **Water Simulation** (with a downward-pointing arrow)
- Water System** section with a dropdown menu currently showing "None". Below the dropdown, the text "No Water System detected" is displayed in red.
- Weather Data Provider** section with a dropdown menu currently showing "Tomorrow.io".
- Tomorrow.io Settings** section containing:
  - API Key**: a text input field.
  - Include Extra Data Package**: a checkbox that is currently unchecked.
  - Global Location** section with two input fields:
    - Latitude**: an input field containing the value "0".
    - Longitude**: an input field containing the value "0".

## TomorrowData

The TomorrowData class is used to store and manage the weather/maritime data obtained from the Tomorrow service.

### TOMORROWDATA CLASS

---

- *latitude* is a float value that represents a geographic coordinate that specifies the north-south position of a point on the Earth's surface. Latitude must be set according to ISO 6709.
- *longitude* is a float value that represents a geographic coordinate that specifies the east-west position of a point on the Earth's surface. Longitude must be set according to ISO 6709.
- *data* is a CoreData class instance that represents the Tomorrow API data.
- *warnings* is a list of TomorrowWarning instances that represent error JSON data.

## TOMORROWDATA CLASS

---

### Weather Data

- *temperature* is a float value that represents the temperature in °C;
- *temperatureApparent* is a float value that represents the temperature equivalent perceived by humans, caused by the combined effects of air temperature, relative humidity, and wind speed. Measured in °C;
- *dewPoint* - the temperature to which air must be cooled to become saturated with water vapor. Measured in °C;
- *humidity* is a float value that represents the concentration of water vapor present in the air. Expressed as a percentage %;
- *windSpeed* - the fundamental atmospheric quantity caused by air moving from high to low pressure, usually due to changes in temperature. Measured in m/s;
- *windDirection* - the direction from which it originates, measured in degrees counter-clockwise from due north, but represented in XY-axis coordinates;
- *windGust* - the maximum brief increase in the speed of the wind, usually less than 20 seconds. Measured in m/s;
- *pressureSurfaceLevel* - the force exerted against a surface by the weight of the air above the surface (at the surface level). Measured in hPa;
- *pressureSeaLevel* - the force exerted against a surface by the weight of the air above the surface (at the mean sea level). Measured in hPa;
- *visibility* - the measure of the distance at which an object or light can be clearly discerned. Measured in km;
- *cloudCover* - the fraction of the sky obscured by clouds when observed from a particular location. Expressed in percentage %;
- *precipitationIntensity* - the amount of precipitation that falls over time, covering the ground in a period of time. Measured in mm/hr;

- *precipitationProbability* - the chance that at least some minimum quantity of precipitation will occur within a specified forecast period and location. Expressed in percentage %.
- *particulateMatter25* - the concentration of atmospheric particulate matter (PM) that have a diameter of fewer than 2.5 micrometers;
- *pollutantO3* - the concentration of surface Ozone (O3). Measured in ppb;
- *pollutantNO2* - the concentration of surface Nitrogen Dioxide (NO2). Measured in ppb;
- *pollutantCO* - the concentration of surface Carbon Monoxide (C). Measured in ppb;
- *pollutantSO2* - the concentration of surface Sulfur Dioxide (SO2). Measured in ppb;
- *treeIndex* - the Tomorrow index representing the extent of grains of overall tree pollen or mold spores in a cubic meter of the air;
- *grassIndex* - the Tomorrow index representing the extent of grains of overall grass pollen or mold spores in a cubic meter of the air;
- *weedIndex* - the Tomorrow index representing the extent of grains of overall weed pollen or mold spores in a cubic meter of the air;
- *precipitationType* - the PrecipitationType enum value that specifies the various types of precipitation that are falling to ground level;

**Weather Code** – the WeatherCode enum value that contains the most prominent weather condition.

```
public enum WeatherCode
{
    [Description("0: Unknown")] Unknown = 0,
    [Description("1000: Clear")] Clear = 1000,
    [Description("1001: Cloudy")] Cloudy = 1001,
    [Description("1100: Mostly Clear")] MostlyClear = 1100,
    [Description("1101: Partly Cloudy")] PartlyCloudy = 1101,
    [Description("1102: Mostly Cloudy")] MostlyCloudy = 1102,
    [Description("2000: Fog")] Fog = 2000,
    [Description("2100: Light Fog")] LightFog = 2100,
    [Description("3000: Light Wind")] LightWind = 3000,
    [Description("3001: Wind")] Wind = 3001,
    [Description("3002: Strong Wind")] StrongWind = 3002,
    [Description("4000: Drizzle")] Drizzle = 4000,
    [Description("4001: Rain")] Rain = 4001,
    [Description("4200: Light Rain")] LightRain = 4200,
    [Description("4201: Heavy Rain")] HeavyRain = 4201,
    [Description("5000: Snow")] Snow = 5000,
    [Description("5001: Flurries")] Flurries = 5001,
    [Description("5100: Light Snow")] LightSnow = 5100,
    [Description("5101: Heavy Snow")] HeavySnow = 5101,
    [Description("6000: Freezing Drizzle")] FreezingDrizzle = 6000,
    [Description("6001: Freezing Rain")] FreezingRain = 6001,
    [Description("6200: Light Freezing Rain")] LightFreezingRain = 6200,
    [Description("6201: Heavy Freezing Rain")] HeavyFreezingRain = 6201,
    [Description("7000: Ice Pellets")] IcePellets = 7000,
    [Description("7101: Heavy Ice Pellets")] HeavyIcePellets = 7101,
    [Description("7102: Light Ice Pellets")] LightIcePellets = 7102,
    [Description("8000: Thunderstorm")] Thunderstorm = 8000
}
```

## Maritime Data

- *waveSignificantHeight* - the height of the combined wind waves and swells. Measured in meters;
- *waveFromDirection* - the direction in which the combined wind waves and swells are moving. Measured in degrees;
- *waveMeanPeriod* - the frequency of the combined wind waves and swells; Measured in seconds;
- *seaSurfaceTemperature* - the temperature of the sea water near the surface (including the part under sea-ice, if any). Measured in °C;
- *primarySwellWaveSignificantHeight* - the height of the primary swell. Measured in meters;
- *primarySwellWaveMeanPeriod* - the frequency of primary swells; Measured in seconds;
- *secondarySwellWaveSignificantHeight* - the height of the secondary swells. Measured in meters;
- *secondarySwellWaveMeanPeriod* - the frequency of secondary swells. Measured in seconds;
- *tertiarySwellWaveSignificantHeight* - the height of the tertiary swells. Measured in meters;
- *tertiarySwellWaveMeanPeriod* - the frequency of tertiary swells. Measured in seconds;
- *tides* - the amplitude of ocean tide measured in meters.

## REQUEST AND ERROR HANDLING

---

The Tomorrow API is organized in a RESTful, stable endpoint structure, administered over HTTPS response codes and authentication. The API has predictable URLs composed of the following main elements:

- *Url*: <https://api.tomorrow.io/v4/timelines>;
- *apikey*;
- *location*: latitude and longitude;
- *fields*: temperature, dewPoint, humidity, and so on;
- *timesteps*: current, 1h, 1d (forecast).
- *endTime*: forecast end time (eg "2022-03-20T14:09:50Z")

## RECEIVE TOMORROW DATA

---

Receiving Tomorrow data can be done by subscribing to the `OnTomorrowDataSent` delegate from the `TomorrowModule` class.

```
public delegate void OnTomorrowDataSent(TomorrowData tomorrowData);  
public OnTomorrowDataSent onTomorrowDataSent;
```

**Example:**

```
_tomorrowModule.onTomorrowDataSent += OnReceivingTomorrowWeatherData;
```

## RECEIVE TOMORROW EXCEPTIONS

---

Receiving Tomorrow exceptions can be done by subscribing to the `OnTomorrowExceptionRaised`.

```
public delegate void OnTomorrowExceptionRaised(string exceptionMessage);  
public OnTomorrowExceptionRaised onTomorrowExceptionRaised;
```

**Example:**

```
_tomorrowModule.onTomorrowExceptionRaised += OnTomorrowExceptionRaised
```

The Tomorrow API uses conventional HTTP response codes to indicate the outcome of an API request. Codes in the 2xx range indicate success, the 4xx category indicates errors in the provided information and 5xx codes imply server error.



## Converting Tomorrow Data

In order to simulate the weather using Tomorrow meteorological data and third-party plugins: Enviro, Tenkoku, and Atmos, or simulate the ocean using the third-party ocean plugins: Crest and KWS, the Tomorrow data must be converted to Real-Time weather data. In other words, the TomorrowData class must be cast to the WeatherData and the WaterData classes.

Data conversion can be performed using the *ConvertCurrentTomorrowDataToRtwData*, *ConvertHourlyTomorrowDataToRtwData* or *ConvertDailyTomorrowDataToRtwData* functions from the TomorrowDataConverter class.

```
TomorrowDataConverter tDC = new  
TomorrowDataConverter(tomorrowData);
```

**Example:**

```
WaterData rtwCurrentWaterData = tDC.ConvertCurrentTomorrowWaterDataToRtwData();  
WeatherData rtwCurrentWeatherData = tDC.ConvertCurrentTomorrowDataToRtwData();  
List<WeatherData> rtwHourlyWeatherData = tDC.ConvertHourlyTomorrowDataToRtwData();  
List<WeatherData> rtwDailyWeatherData = tDC.ConvertDailyTomorrowDataToRtwData();
```

Based on the timestep, *ConvertCurrentTomorrowDataToRtwData* will convert only *current* timestep data and return WeatherData, and *ConvertHourlyTomorrowDataToRtwData*/  
*ConvertDailyTomorrowDataToRtwData* will convert only *1h/1d* timestep data and return a WeatherData list.

Due to differences in the weather data structure, the IndexUV will have a default value after conversion. Once the data conversion is complete, the notification function can be invoked to update the system weather data.

**Example:**

```
NotifyCurrentWaterChanged(rtwWaterData);  
NotifyCurrentWeatherChanged(rtwWeatherData);  
NotifyHourlyWeatherChanged(rtwWeatherDataList);  
NotifyDailyWeatherChanged(rtwWeatherDataList);
```

# MetOcean



## About

The MetOcean Solutions Point Forecast API [\[Link\]](#) provides forecast and recent historical ocean, terrestrial and atmospheric data. (© MetOcean Solutions)

In order to obtain data from the MetOcean API, an API key [\[Link\]](#) should be introduced as a request header. The requested data is returned through a JSON-encoded response.

Access to MetOcean weather and ocean data requires a valid API key and proper location information.

### REQUEST METHODS

—

To request weather and ocean data from MetOcean, you must configure the following settings in the MetOcean Module inspector:

- Specify a valid *API key* in the MetOcean API key field.
- Specify the *latitude* and *longitude* (ISO 6709).
- The interval in hours of the requested data.

### PARAMETERS

—

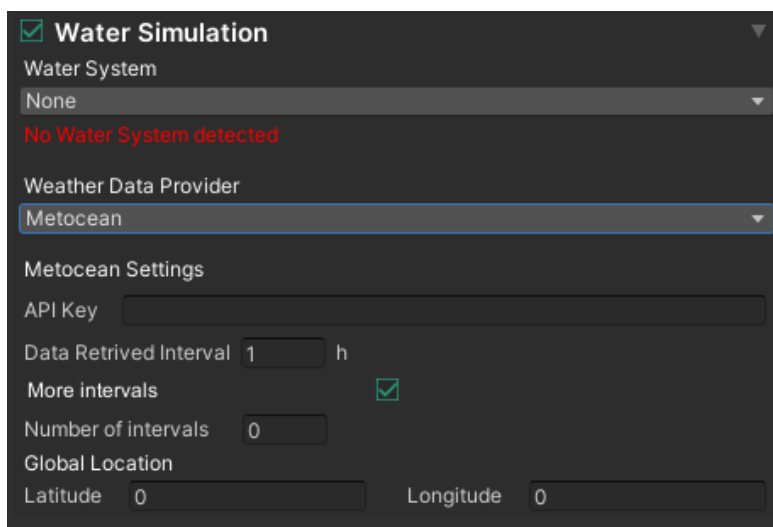
The MetOcean API will automatically request the core weather/maritime information.

All the available variables can be seen here: [\[Link\]](#).

## Inspector interface

The Module inspector contains three main options:

- *API Settings* – access key to MetOcean data;
- *Location Settings* – settings for the location from where the weather/ocean information will be requested;
- *Parameter Settings* – settings for the data returned in intervals (the default, interval – 1, repeat – 0, only returns the current data).

The image shows a dark-themed inspector interface for a module. At the top, there is a checked checkbox labeled 'Water Simulation'. Below it is a 'Water System' dropdown menu currently set to 'None', with a red message 'No Water System detected' below it. The 'Weather Data Provider' dropdown menu is set to 'Metoccean'. Under 'Metoccean Settings', there is an 'API Key' text field. Below that is 'Data Retrived Interval' set to '1' with a unit 'h'. Next to it is a 'More intervals' checkbox which is checked. Below that is 'Number of intervals' set to '0'. At the bottom, there is a 'Global Location' section with 'Latitude' and 'Longitude' text fields, both currently set to '0'.

Interval In Hours – forecast data will be returned at an interval of 1 hour by default;

Repeat – how much forecast data should be obtained, (the default is 0, meaning that the module will only get the current data); if the repeat is set to a higher number than 0, multiple data will be returned in the same point at the set interval of hours.

## MetOcean Data

The `MetoceanData` class is used to store and manage the data introduced in the request or obtained from the Metocean API service.

### MetoceanData CLASS

---

- *Latitude* is a float value that represents a geographic coordinate that specifies the north-south position of a point on the Earth's surface. Latitude must be set according to ISO 6709.
- *Longitude* is a float value that represents a geographic coordinate that specifies the east-west position of a point on the Earth's surface. Longitude must be set according to ISO 6709.
- *TimeMetocean* contains a list of `DateTime` values for all the returned variables.
- *NoDataReasons* - used for notifying the user why information at a specific point could not be returned.

### Variables class

---

- *AirTemperature* contains a list of values representing the temperature in °C;
- *PrecipitationRate* contains a list of values representing the amount of precipitation that falls over time measured in mm/hr;
- *AirPressureAtSea* contains a list of values representing the force exerted against a surface by the weight of the air above the surface (at the mean sea level), measured in Pa;
- *WindDirection* contains a list of values representing the direction from which the wind originates, measured in degrees counter-clockwise from due north;

- *WindSpeed* contains a list of values representing the velocity of the wind at the specified point measured in m/s;
- *AirHumidity* contains a list of values representing the concentration of water vapor present in the air measured in percentages;
- *AirVisibility* contains a list of values representing the measure of the distance at which an object or light can be clearly discerned, measured in meters;
- *WaveHeight* contains a list of values representing the vertical distance between the crest (peak) and the trough of a wave, measured in meters;
- *WavePeriodPeak* contains a list of values representing the time it takes for two successive wave crests to reach a fixed point, measured in seconds;
- *WaveDirection* contains a list of values representing the direction of waves in degrees, with 0 degrees being North;
- *CloudCover* contains a list of values representing the fraction of the sky obscured by clouds when observed from a particular location, measured in percentages;
- *RadiationFluxLongwave* contains a list of values representing a product of both downwelling infrared energy as well as emission by the underlying surface, measured in  $W/m^2$ ;
- *RadiationFluxShortwave* contains a list of values representing a result of specular and diffuse reflection of incident shortwave radiation by the underlying surface, measured in  $W/m^2$ ;

## RECEIVE METOCEAN DATA

The API delivers atmospheric and maritime data from global forecast models. (© MetOcean Solutions)

The address of the resource to retrieve via HTTP GET consists of:

- *Base Url:* <https://forecastv2.metoceanapi.com/point/time;>
- *API Key;*
- *location:* latitude and longitude;
- *variables;*
- *interval;*
- *repeat;*

Receiving MetOcean data can be done by subscribing to the `OnMetoceanDataSent` delegate from the `MetoceanModule` class.

```
public delegate void OnMetoceanDataSent(MetoceanData metoceanData);  
public OnMetoceanDataSent onMetoceanDataSent;
```

**Example :**

```
_metoceanModule.onMetoceanDataSent += OnReceivingMetoceanData;
```

## RECEIVE METOCEAN EXCEPTIONS

Receiving MetOcean exceptions can be done by subscribing to the `OnMetoceanExceptionRaised`.

```
public delegate void OnMetoceanExceptionRaised(ExceptionType exception, string  
exceptionMessage);  
public OnMetoceanExceptionRaised onMetoceanExceptionRaised;
```

**Example :**

```
_metoceanModule.onMetoceanExceptionRaised += OnMetoceanExceptionRaised;
```

The MetOcean API uses conventional HTTP response codes to indicate the outcome of an API request. Codes in the 2xx range indicate success, the 4xx category indicates errors in the provided information and 5xx codes imply server error.

## Converting Metocean Data

Converting MetOcean data consists of translating the information from the API into the WaterData structure of Real-Time Weather, in order for it to be later used in simulating the ocean and weather conditions.

Data conversion can be performed using the *ConvertCurrentMetoceanDataToRtwData* or *ConvertIntervalMetoceanDataToRtwData* functions from the MetoceanDataConverter class.

```
MetoceanDataConverter metoceanDataConverter = new  
MetoceanDataConverter(metoceanData);
```

**Example :**

```
WaterData rtwCurrentWathrData =  
MetoceanDataConverter.ConvertCurrentMetoceanDataToRtwData();  
List<WaterData> rtwHourlyWaterData =  
MetoceanDataConverter.ConvertIntervalMetoceanDataToRtwData();
```

The *ConvertCurrentMetoceanDataToRtwData* method will only create one WaterData instance, while the *ConvertIntervalMetoceanDataToRtwData* method will return a list of WaterData structures that will save forecast information.

Once the data conversion is complete, the notification function can be invoked to update the system water data.

**Example :**

```
NotifyCurrentWaterDataChanged(rtwWaterData);  
NotifyIntervalWaterDataChanged(rtwWaterDataList);
```

## Stormglass



## About

The Storm Glass API provides high-resolution forecasts for up to 10 days ahead as well as historical data. Marine data including tide data is available for all oceans and seas worldwide. (©Storm Glass AB)

## REQUEST METHODS

—

To request maritime and weather data from Stormglass, you must configure the following settings in the Stormglass Module inspector:

- Specify a valid *API key* in the Stormglass API key field.
- Specify *latitude* and *longitude* (ISO 6709).

## PARAMETERS

—

The Stormglass API will automatically request the core weather and maritime data such as temperature, wind speed and direction, wave height, and so on. In the "Extra Options" menu, check what forecast information and what additional information you want to request from the API. You can choose from a set of weather parameters related to the chemical composition of the water, such as iron amount, chlorophyll, Ph, and so on.



## Inspector interface

The Stormglass Module inspector contains three main options:

- *API Settings* - Stormglass API settings, such as access key;
- *Position Options* - settings for the location from where the weather and maritime information will be requested;
- *Extra Options* - settings that define what weather and maritime parameters will be requested from the API.

Additional parameters that provide tide or biological information, must be checked to be requested from the API. Otherwise, these parameters will have default values.

## Stormglass Data

The Stormglass Data classes are used to store and manage the weather and maritime data obtained from the Stormglass service.

### StormglassWeatherData

Stores the maritime and weather data such as:

- *AirTemperature*: Represents the air temperature in °C
- *Pressure*: Represents the air pressure in hPa
- *CloudCover*: Represents the total cloud coverage expressed as a percentage
- *currentDirection*: Represents the direction of current. 0° indicates current coming from north
- *currentSpeed*: Represents the speed of current in meters per second
- *gust*: Represents the wind gust in meters per second
- *humidity*: Represents the relative humidity expressed as a percentage
- *precipitation*: Represents the main precipitation in  $\text{kg/m}^2/\text{h} = \text{mm/h}$
- *snowDepth*: Represents the depth of snow in meters
- *visibility*: Represents the horizontal visibility in km
- *iceCover*: Represents the proportion, 0-1
- *seaLevel*: Represents the sea level relative to MSL

- *swellDirection*: Represents the direction of swell waves. 0° indicates swell coming from north
- *secondarySwellDirection*: Represents the direction of secondary swell waves. 0° indicates swell coming from north
- *swellHeight*: Represents the height of swell waves in meters
- *secondarySwellHeight*: Represents the height of secondary swell waves in meters
- *swellPeriod*: Represents the period of swell waves in seconds
- *secondarySwellPeriod*: Represents the period of secondary swell waves in seconds
- *waterTemperature*: Represents the water temperature in degrees Celsius
- *waveDirection*: Represents the direction of combined wind and swell waves. 0° indicates waves coming from north
- *waveHeight*: Represents the significant height of combined wind and swell waves in meters
- *wavePeriod*: Represents the period of combined wind and swell waves in seconds
- *windWaveDirection*: Represents the direction of wind waves. 0° indicates waves coming from north
- *windWaveHeight*: Represents the height of wind waves in meters
- *windWavePeriod*: Represents the period of wind waves in seconds
- *windDirection*: Represents the direction of wind at 10m above sea level. 0° indicates wind coming from north
- *windSpeed*: Represents the speed of wind at 10m above sea level in meters per second.

## StormglassBioData

Stores the biological data such as:

- *chlorophyll*: Represents the mass concentration of chlorophyll-a in sea water
- *iron*: Represents the mole concentration of dissolved iron in sea water
- *nitrate*: Represents the mole concentration of nitrate in sea water
- *phyto*: Represents the net primary production of biomass expressed as carbon per unit volume in sea water
- *oxygen*: Represents the mole concentration of dissolved molecular oxygen in sea water
- *ph*: Represents the sea water pH reported on total scale
- *phytoplankton*: Represents the mole concentration of phytoplankton expressed as carbon in sea water
- *phosphate*: Represents the mole concentration of phosphate in sea water
- *silicate*: Represents the mole concentration of silicate in sea water
- *salinity*: Represents the sea water salinity given in per mil
- *soilMoisture*: Represents the volumetric soil moisture content at 0 to 10 cm below surface
- *soilTemperature*: Represents the soil temperature at 0 to 10 cm below surface

## StormglassTideData

Stores the tide data such as:

- *height*: Represents the height in meters
- *type*: Represents the type of extreme. Either low or high

## StormglassProviders

Stores the data from all the available stormglass data sources:

- *icon*: Represents Germany's National Meteorological Service, the Deutscher Wetterdienst
- *noaa*: Represents the National Oceanic and Atmospheric Administration
- *meteo*: Represents the French National Meteorological service
- *dwd*: Represents Germany's National Meteorological Service, the Deutscher Wetterdienst
- *meto*: Represents the United Kingdom's national weather service, The UK MetOffice
- *fcoo*: Represents the Danish Defence Centre for Operational Oceanography
- *fmi*: Represents the Finnish Meteorological Institution
- *yr*: Represents the Norwegian Meteorological Institute and NRK
- *smhi*: Represents the Swedish Meteorological and Hydrological Institute
- *sg*: Represents the Stormglass Ai
- *mercator*: For bio attributes

## StormglassRequestData

Stores data about the API request:

- *cost*: Represents the cost of the request
- *dailyQuota*: Represents the number of requests available for one day
- *end*: Represents the end date of the forecast data
- *lat*: Represents the latitude of the requested data
- *lng*: Represents the longitude of the requested data
- *requestCount*: Represents the request number for the current day
- *start*: Represents the start date of the forecast data

## RECEIVE STORMGLASS DATA

Receiving Stormglass data can be done by subscribing to the `OnWaterDataReceived` or `OnWeatherDataReceiveds` delegate from the `StormglassModule` class.

```
public delegate void OnWaterDataReceived(StormglassWeatherData weatherData);  
public delegate void OnWeatherDataReceived(StormglassWeatherParams weatherData);  
  
public OnWeatherDataReceived OnWeatherDataReceived;  
public OnWaterDataReceived OnWaterDataReceived;
```

### Example :

```
_stormglassModule.OnWeatherDataReceived += OnReceivingStormglassWeatherData;  
_stormglassModule.OnWaterDataReceived += OnReceivingStormglassWaterData;
```

## RECEIVE STORMGLASS EXCEPTIONS

Receiving Stormglass exceptions can be done by subscribing to the `OnStormglassExceptionRaised`.

```
public delegate void OnStormglassExceptionRaised(ExceptionType exception, string  
exceptionMessage);  
public OnStormglassExceptionRaised OnStormglassExceptionRaised;
```

**Example:**

```
_stormglassModule.OnStormglassExceptionRaised += OnStormglassExceptionRaised
```

## Converting Stormglass Data

In order to simulate the weather using Stormglass weather and maritime data with the third-party plugins, the Stormglass data must be converted to Real-Time weather data using the `WeatherData` class respectively `WaterData` class.

Data conversion can be performed using the `ConvertCurrentStormglassDataToWeatherData`, `ConvertHourlyStormglassDataToWeatherData`, `ConvertCurrentStormglassDataToWaterData` or `ConvertHourlyStormglassDataToWaterData` methods from the `StormglassDataConverter` class.

```
StormglassDataConverter dataConverter =  
new TomorrowDataConverter();
```

**Example:**

```
WeatherData rtwCurrentWeatherData =  
dataConverter.ConvertCurrentStormglassDataToWaterData(stormglassData);
```

```
List<WeatherData> rtwHourlyWeatherData =  
dataConverter.ConvertHourlyStormglassDataToWaterData(stormglassData);
```

```
WaterData rtwCurrentWaterData =  
dataConverter.ConvertCurrentStormglassDataToWeatherData(stormglassData);
```

```
List<WaterData> rtwHourlyWaterData =  
dataConverter.ConvertHourlyStormglassDataToWeatherData(stormglassData);
```

# Reverse Geocoding

## About

Reverse geocoding is the process of converting geographic coordinates (*latitude, longitude*) to precise locality information.

Real-Time Weather uses *Nominatim*, a free API tool that generates address names based on ISO 6709 coordinates through Reverse Geocoding. More information about this API is available here: <https://nominatim.org/release-docs/latest/>.

## REQUEST METHODS

To use the reverse geocoding functionality, a coroutine must be created, as in the example below, which calls the `RequestGeocodingInformation` function from the `ReverseGeocoding` class.

```
private IEnumerator GetGeocodingInformation(float latitude, float longitude)
{
    ReverseGeocoding reverseGeocoding = new ReverseGeocoding();
    CoroutineWithData reverseGeoCoroutine = new CoroutineWithData(this,
        reverseGeocoding.RequestGeocodingInformation(latitude, longitude));

    yield return reverseGeoCoroutine.Coroutine;

    GeocodingData reverseGeoData +
    (GeocodingData)reverseGeoCoroutine.Result;

    if (reverseGeoData != null)
    {
        Debug.Log("City=" + reverseGeoData.Address.City);
        Debug.Log("Country="+reverseGeoData.Address.Country);
    }
}
```



The result of the reverse geocoding function is a `GeocodingData` object that contains precise address data such as the road, neighborhood, city, country name, and so on.

## GeocodingData

The `GeocodingData` class is used to store and manage the geocoding information obtained from the Nominatim Reverse Geocoding API.

- *latitude* is a float value that represents a geographic coordinate that specifies the north–south position of a point on the Earth's surface. Latitude must be set according to ISO 6709.
- *longitude* is a float value that represents a geographic coordinate that specifies the east–west position of a point on the Earth's surface. Longitude must be set according to ISO 6709.
- *address* is an object that contains every bit of locality information in the English language:
  - *neighborhood* is a string value that represents the localized neighbourhood (mostly returned in cities).
  - *suburb* is a string value that represents the located city suburb (or neighbourhood).
  - *municipality* is a string value that represents the localized municipality (not necessarily a city).
  - *village/town/city* is a string value that represents the localized village/town/city name.
  - *county/district* is a string value that represents the localized county/district name (could be shown if no locality is found).
  - *state* is a string value that represents the localized state name.
  - *country* is a string value that represents the localized country name.
  - *countryCode* is a string value that represents the country code as defined by ISO 3166-1 standard.

# Custom Forecast Window

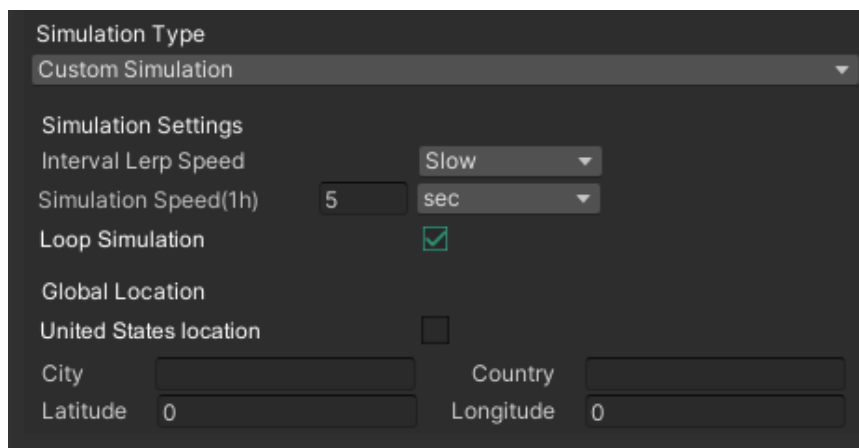
The main functionality of this feature is that you can create a complete customizable forecast where you can specify the weather state for every hour interval set.

The forecast preset window contains the following options:

## General data fields

Every preset contains global data used for the entire forecast simulation.

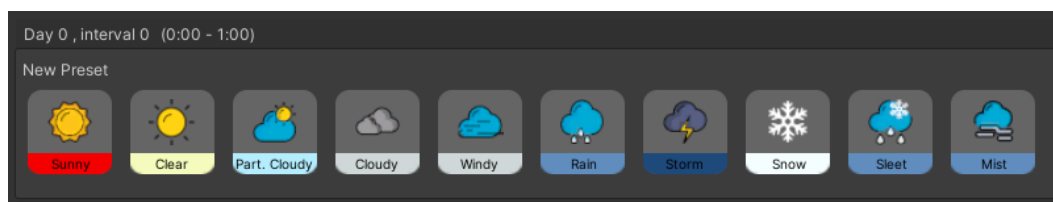
- **Interval Lerp Speed:** The transition speed between weather intervals
- **Simulation Speed:** How much an hour of the forecast simulation represents in real time.
- **Loop Forecast:** If checked, the forecast will loop through the days
- **Location:** The location of the current simulation is based on latitude and longitude or city and country.



The screenshot shows a settings window for a custom simulation. It includes a dropdown for 'Simulation Type' set to 'Custom Simulation'. Under 'Simulation Settings', there are fields for 'Interval Lerp Speed' (set to 'Slow'), 'Simulation Speed(1h)' (set to '5' with a unit of 'sec'), and a checked 'Loop Simulation' checkbox. The 'Global Location' section has an unchecked 'United States location' checkbox. Below this are input fields for 'City', 'Country', 'Latitude' (set to '0'), and 'Longitude' (set to '0').

# Weather Presets

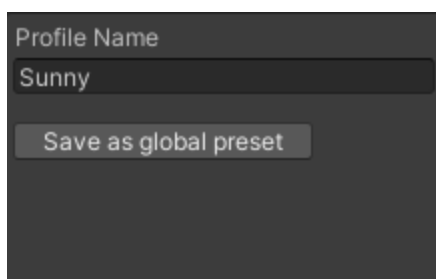
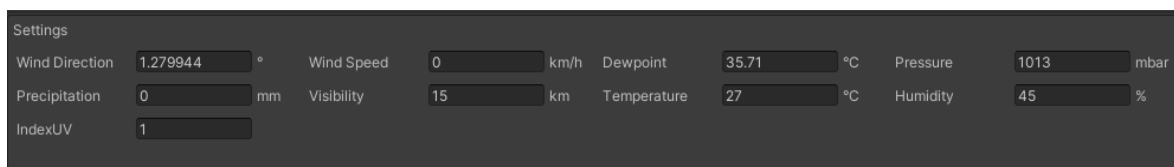
The weather presets are used to store weather data that can be applied to specific intervals. After clicking on an interval, you can select one of the presets to be applied to that interval.



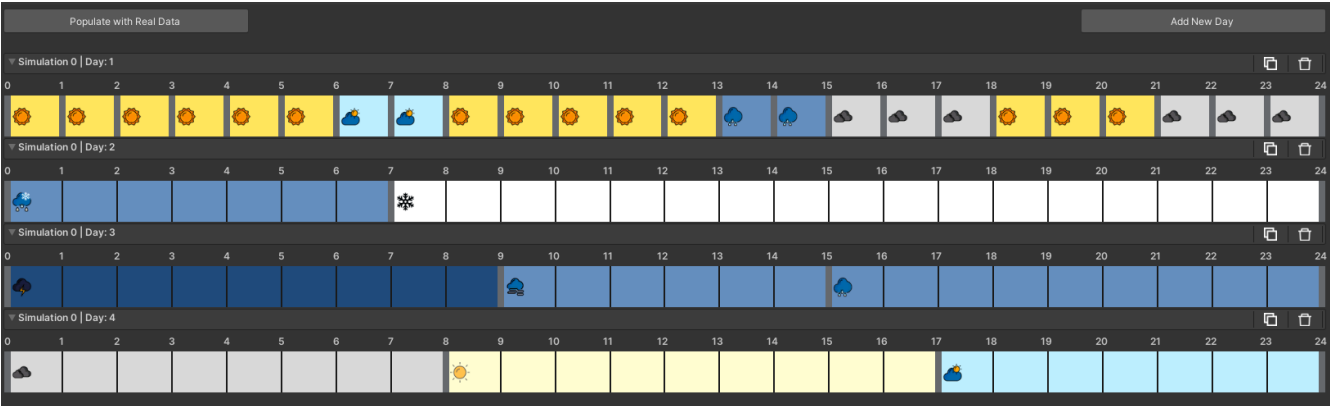
To update the name of an already existing preset, select the preset from the list and modify its name.

All weather data for the selected interval can be edited.

Pressing the "Save as global preset" button will set the default preset as the selected preset. And when creating new intervals, they will be populated with that data.



# Day data fields

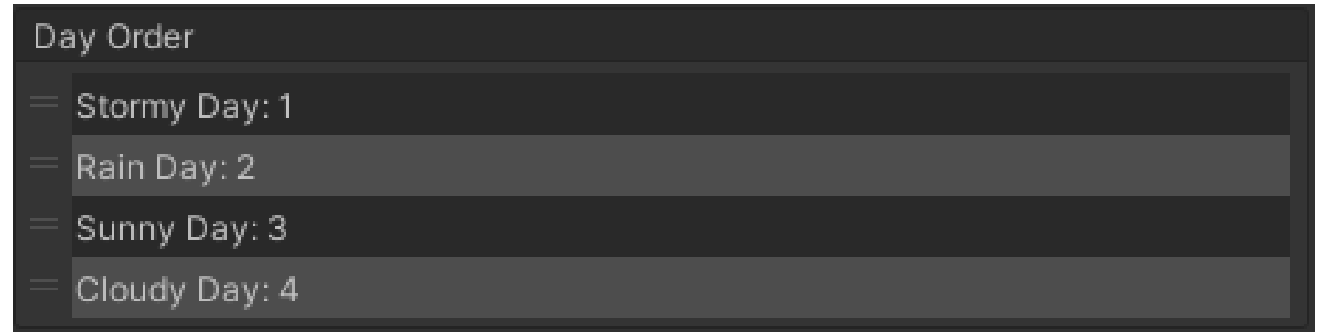


For each day, you will be able to specify up to 24 different hour intervals. Each interval will have its own weather data where you will be able to modify the weather parameters. To add a new day, press the "Add New Day" button.

Each day has a name field where you can set a custom name.

You can duplicate or delete a specific day configuration by pressing the buttons at the top right of the tab.

# Reordering Days



To reorder the days, simply select the day from the list on the left and drag it to your desired location.

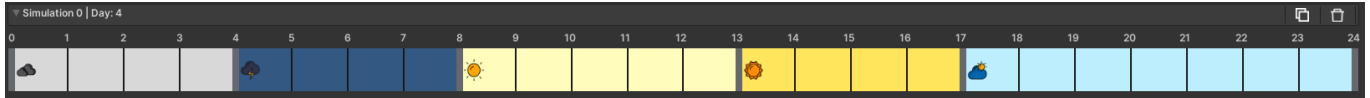


## Selecting Intervals



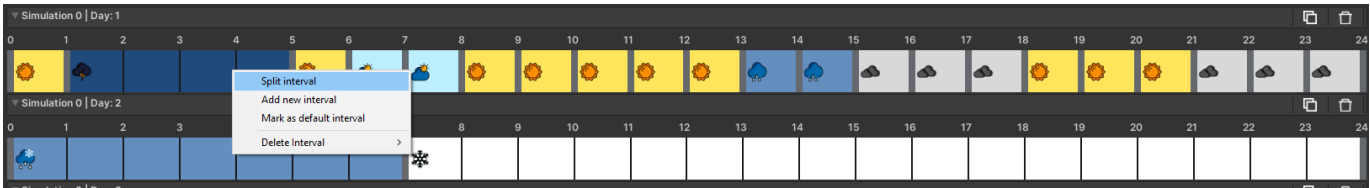
If you left-click an interval from any day, it will get selected, and the background color will turn lighter to highlight it.

## Mark As Default Interval



To mark an interval as default, right click it and select "Mark as default interval". It will be filled with default interval data. If you change the data from a default interval, it will become a custom interval.

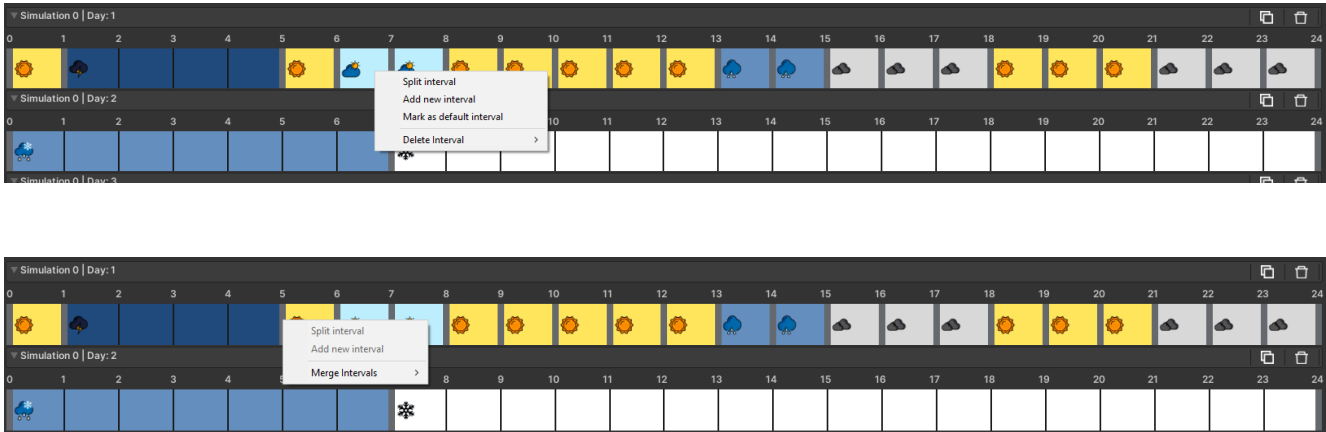
## Adding New Intervals/ Splitting Intervals



To add a new interval, right-click an interval then select the "Add new interval" option. A new one-hour interval will be created that will have the default interval data.

If you select to split an interval, then a new interval delimiter will appear at the mouse position, and both new intervals will have the same data as the original interval.

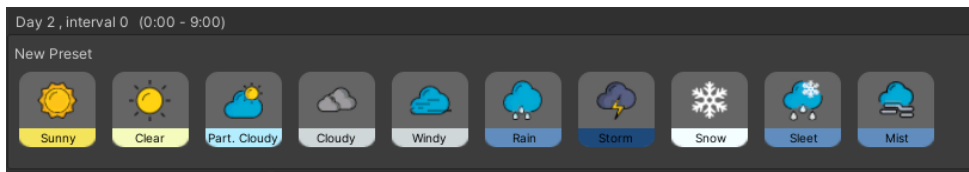
# Deleting Intervals/ Merging Intervals



To delete an interval, right-click it and then select "Delete interval". Then you will have to choose which one of the neighboring intervals will take the space of the deleted interval.

If you right-click on an interval delimiter, you will be prompted with the option of merging the intervals; you will also need to choose which interval will be kept.

## Preset Tab



The preset tab will have listed all the presets currently present in the project. To assign a preset to an interval, simply left-click it while having the interval selected.

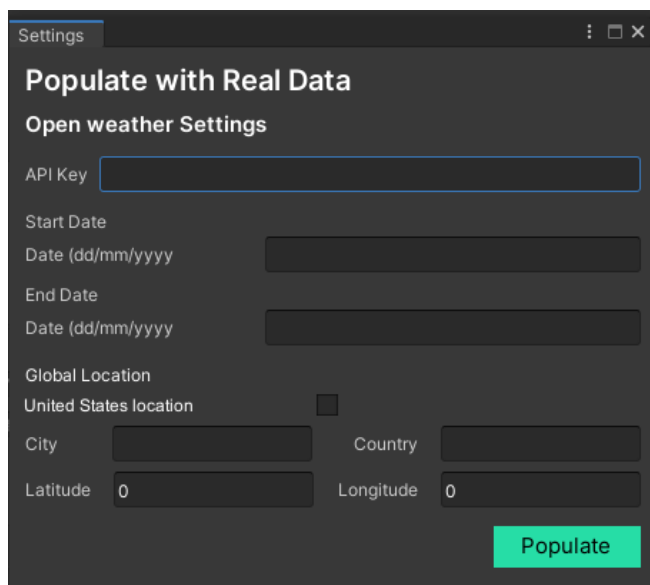
# Save Providers Forecast

The main functionality of this feature is that you can save the forecast data from providers to a timelapse that could be reused and edited.

## Forecast saving

Pressing the "Populate with real data" button will open a tab to request historical weather data. After completing the required data, hit the "Populate" button. After that the simulation data will be overwritten by the requested data.

This requires an OpenWeather Api key that comes with a historical data option.

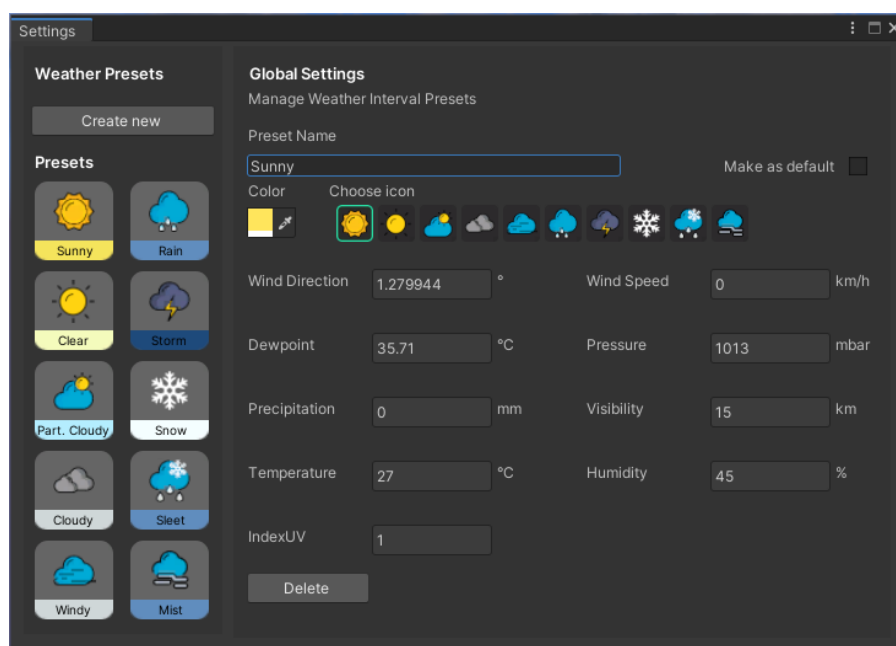
A screenshot of a software window titled "Settings" with a sub-tab "Populate with Real Data". The window contains a section titled "Open weather Settings" with several input fields: "API Key" (a single-line text box), "Start Date" with a label "Date (dd/mm/yyyy)" and a text box, "End Date" with a label "Date (dd/mm/yyyy)" and a text box, "Global Location" with a checked checkbox, "United States location" with an unchecked checkbox, "City" and "Country" (two side-by-side text boxes), and "Latitude" and "Longitude" (two side-by-side text boxes, both containing the value "0"). A green "Populate" button is located at the bottom right of the form.

## Edit the forecasts

After the data is received, you can edit the forecast like other custom simulations.

## Create new forecast presets

Pressing the "Settings" button on the RealTimeWeather manager panel will open the tab where you can create and edit weather presets.



Pressing "Create new" will add the new weather preset to the list.

After selecting one of the presets by clicking on it, you can edit all weather data, select a custom color, give a custom name, and select the image.

The "Make as default" toggle will make the selected preset to be used as default data in the weather editor.



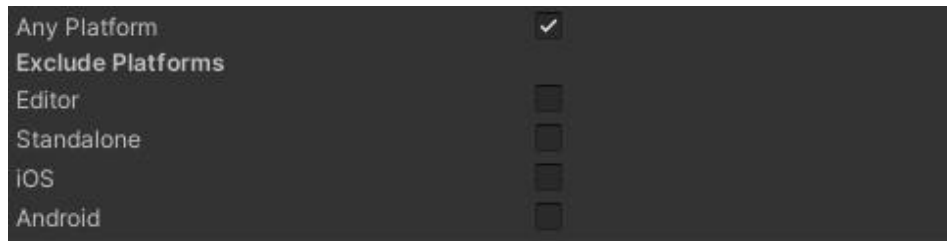
# Platform Compatibility

## Real-Time Weather Manager DLL

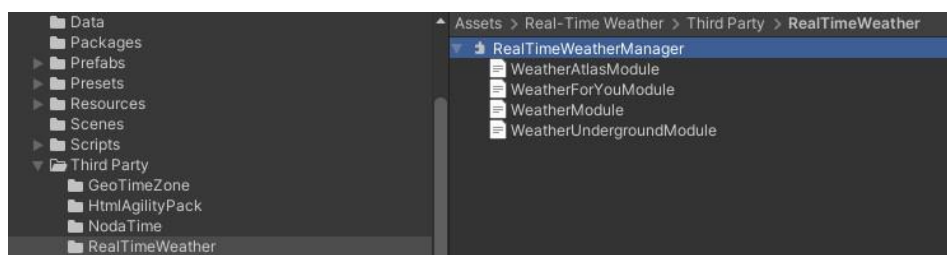
Real-Time Weather is a stable solution for the following platforms:

- Windows
- MacOS
- Linux/Ubuntu (UNIX)
- Android
- iOS

Only one Real-Time Manager DLL exists in the project, that supports all mentioned platforms.



The DLL can be found in “Real-Time Weather\Third Party\RealTimeWeather\” path.



# URP/HDRP Compatibility

## Plugin Compatibility Information

Currently integrated plugins URP/HDRP support status:

- **Enviro**
  - Compatible with both URP/HDRP 7.5+;
  - Scriptable RPs supported from Unity version 2019.4.26f1;
- **Tenkoku**
  - Only compatible with Standard RP;
  - Scriptable RPs (URP/HDRP) are NOT currently supported;
- **Massive Clouds – Atmos**
  - Standard RP supported from Unity version 2018.4+;
  - URP/HDRP supported from 2019.3+;
  - VR related information:
    - URP: only Single Pass Rendering supported;
    - HDRP: NOT currently supported for VR;
- **Expanse**
  - HDRP supported from Unity version 2020.1.17+;

Unsupported plugins will be signalled with a warning message in the Inspector panel:



Enviro  
Enviro not found in your project!