REAL-TIME WEATHER

1.5.0

—

Documentation

—

Thank you for your purchase!

ASSIST

TABLE OF CONTENTS

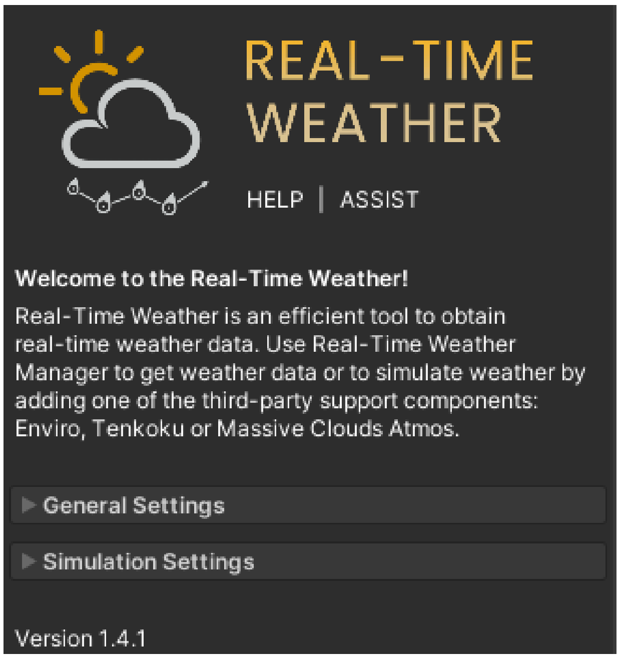
INSTALLATION

Import 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.



Welcome to the Real-Time Weather!

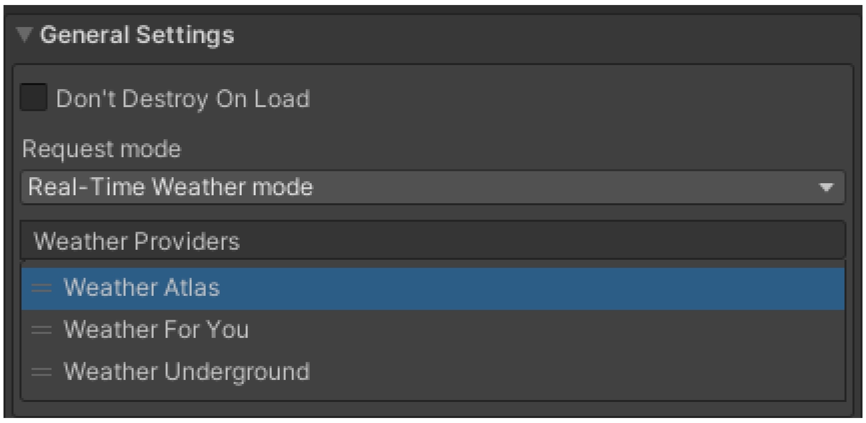
Real-Time Weather is an efficient tool to obtain real-time weather data. Use Real-Time Weather Manager to get weather data or to simulate weather by adding one of the third-party support components: Enviro, Tenkoku or Massive Clouds Atmos.

Version 1.4.1

STEP 2

—

If you want the “Real-time Weather Manager” to not be destroyed when other scenes are loaded, then check “Don’t Destroy On Load” in the “General Settings” panel.



General Settings

Don't Destroy On Load

Request mode: Real-Time Weather mode

Weather Providers

- Weather Atlas
- Weather For You
- Weather Underground

STEP 3

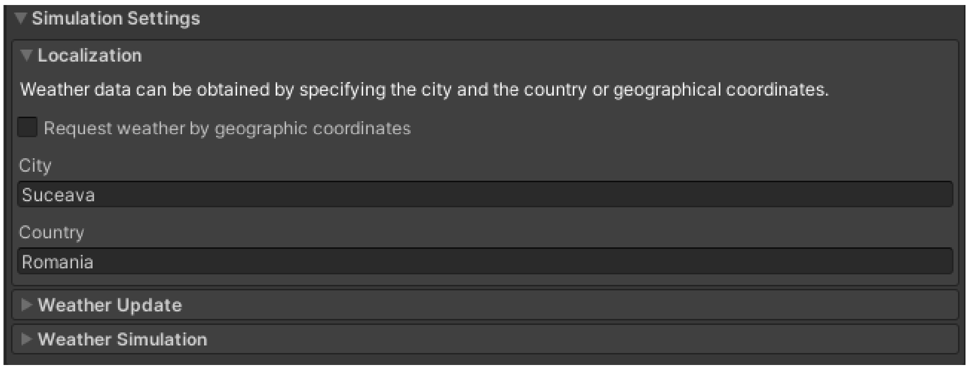
—

For Real-Time Weather mode there are 3 sources for weather information in case one does not work. Also, the user can choose the order of the weather providers on the list from which the data is requested, so that a provider can have a higher priority than others.

STEP 4

—

In the “Localization” area fill up the “City” and “Country” input fields for the location or use geographical coordinates. For USA, the “City” input should always be composed of two words, the name of the state and the name of the city, for example: "Colorado, Arriba".



Simulation Settings

Localization

Weather data can be obtained by specifying the city and the country or geographical coordinates.

Request weather by geographic coordinates

City: Suceava

Country: Romania

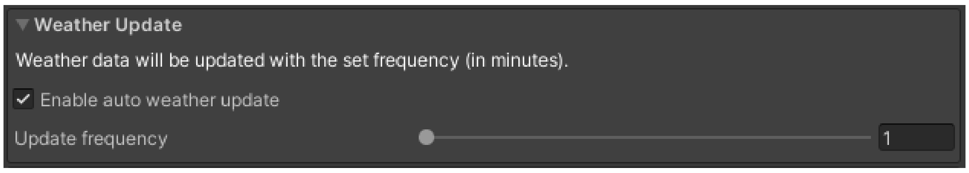
Weather Update

Weather Simulation

STEP 5

—

To get real-time weather updates, check “Enable auto weather update” and also choose “update frequency” which is measured in minutes.



Weather Update

Weather data will be updated with the set frequency (in minutes).

☑ Enable auto weather update

Update frequency: 1

STEP 6

—

Depending on what weather plugin you have in the project, you can activate either the Enviro, Tenkoku Simulation or Atmos. Simply click on “Activate Enviro Simulation”, “Activate Tenkoku Simulation” or "Activate Atmos Simulation", everything that you need for the weather simulation to run into your scene will be added.

NOTE: For the 2017 and 2018 Unity versions, you should choose to build using IL2CPP when developing for the Android platform. *(Player Settings -> Other Settings -> Scripting Backend -> IL2CPP)*

ENVIRO INTEGRATION

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.

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 SYSTEM

TENKOKU INTEGRATION

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 with wonderful results.

The current Real-Time Weather version supports integration with Tenkoku, that 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- VOLUMETRIC SKYBOX

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.

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 make the settings shown in the Atmos Settings section.

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 made:

ATMOS SETUP WIZARD

—

Lunch the Setup Wizard from 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 *Fix Now* button. The setup wizard supports the appropriate setup for each pipeline. Thus, select the Renderer according. For **Standalone** select *Physics StandardRP - Natural - High* and for the **Android** platform select *Physics StandardRP - Natural - Middle.*

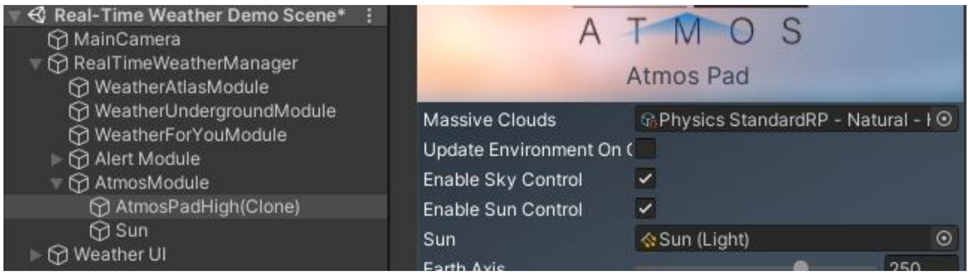
For the Android platform some optimizations can be done that are described in the Atmos plugin's documentation, available here:

[*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.



```
Real-Time Weather Demo Scene*
  MainCamera
  RealTimeWeatherManager
    WeatherAtlasModule
    WeatherUndergroundModule
    WeatherForYouModule
    Alert Module
    AtmosModule
      AtmosPadHigh(Clone)
      Sun
  Weather UI
```

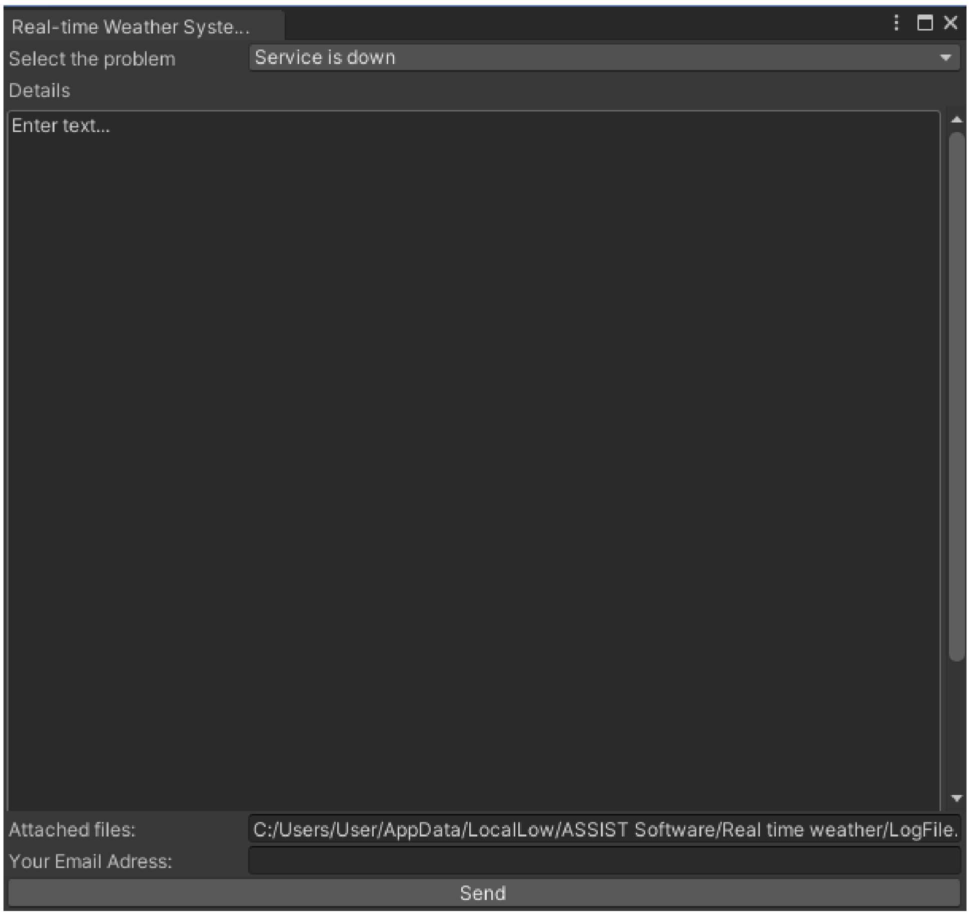
```
Atmos Pad
Massive Clouds            Physics StandardRP - Natural - I
Update Environment On (   
Enable Sky Control        ✓
Enable Sun Control        ✓
Sun                       Sun (Light)
Earth Axis                250
```

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.

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 whenever the *Weather For You* module fails to obtain weather data / whenever a service fails to obtain weather data. It can also be activated by going to the "*Real-time Weather Manager*" tab and selecting "*Bug Reporter*".


Real-time Weather Syste...
Select the problem: Service is down
Details
Enter text...
Attached files: C:/Users/User/AppData/LocalLow/ASSIST Software/Real time weather/LogFile.
Your Email Adress:
Send

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 module, Underground module and Weather For You module, and it also manages the automatic weather data update and weather simulation using third-party support components: Enviro, Tenkoku and Massive Clouds Atmos.

Request weather data

—

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

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

Example:
RealTimeWeatherManager.instance.RequestWeather("Paris", "France");
```

Receive weather data

—

Receiving current weather data can be done by subscribing to the *OnCurrentWeatherUpdate* event. To receive forecast weather (hourly or daily), 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, OnDailyWeatherUpdate* events.

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

WEATHER DATA

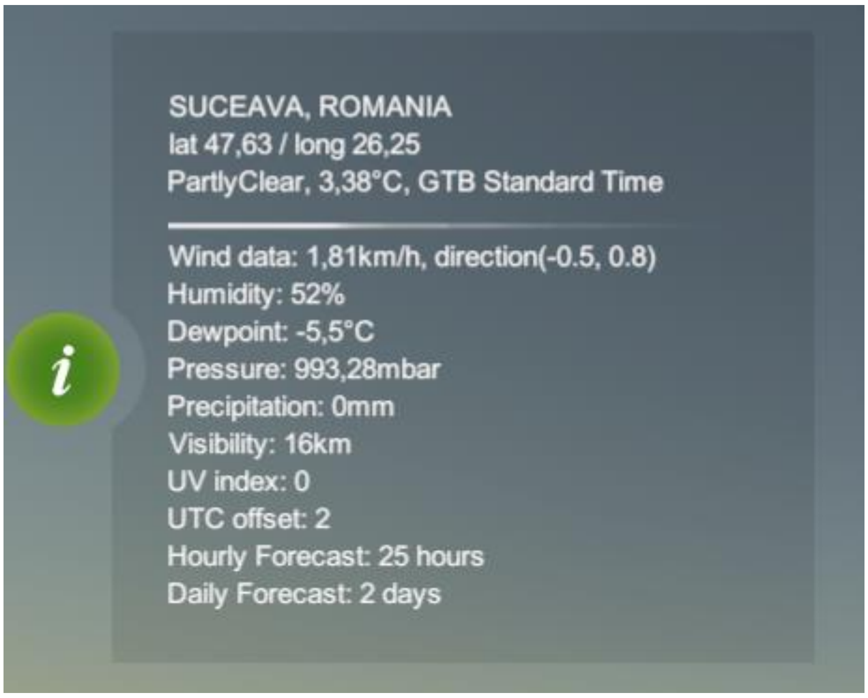
The WeatherData class is used to store and manage weather data.

- *`Localization`* is a Localization class instance that holds the localization data: city, country, latitude, and longitude.
- *`DateTime`* is an instance of 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.
- *`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,
    }
```

- *`Temperature`* is a float value that represents the temperature in °C.
- *`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 following format "Continent/Country".
- *`Precipitation`* is a float value that represents the precipitation in mm.
- *`Dewpoint`* is the temperature to which air must be cooled to become saturated with water vapor in °C.
- *`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.


SUCEAVA, ROMANIA
lat 47,63 / long 26,25
PartlyClear, 3,38°C, GTB Standard Time

Wind data: 1,81km/h, direction(-0.5, 0.8)
Humidity: 52%
Dewpoint: -5,5°C
Pressure: 993,28mbar
Precipitation: 0mm
Visibility: 16km
UV index: 0
UTC offset: 2
Hourly Forecast: 25 hours
Daily Forecast: 2 days

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 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 exception

—

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

OpenWeather

—

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.

OPEN WEATHER MAP SERVICE

ABOUT

OpenWeatherMap [Link] is a paid service which offers the opportunity to obtain real-time weather data through miscellaneous HTTP requests to their server.

The server can be requested through more [APIs] using the API key specific to your account. Real-Time Weather Manager uses the [Current Weather Data API] and [One Call API API].

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

Current weather

Uses this link :
`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 US) and country code.
- City ID. [cities.json]
- Latitude & longitude.
- Zip code and country code.

PARAMETERS

There are 3 type of parameters:

- Language: en, fr, de, jp, ro, etc.
- Units: standard, metric or imperial.
- Request mode represents those 4 types of requests presented.

Weather forecast for a period of time

For Geographic Coordinates Request mode, the weather forecat data can be requested for the next 48 hours with a step of one hour or for the next 7 days with a daily forecast.

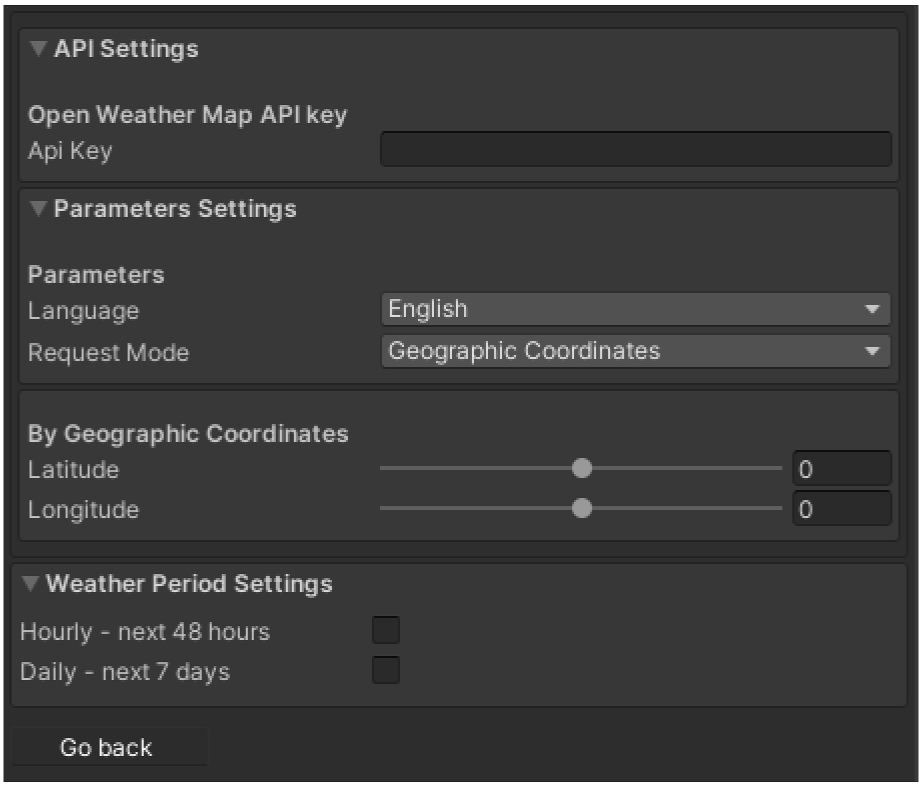
It is used the following link:

```
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 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 for the next 48 hours is wanted, the exclude parameter will also contain the **daily** value.

INSPECTOR INTERFACE


API Settings

Open Weather Map API key
Api Key

Parameters Settings

Parameters
Language: English
Request Mode: Geographic Coordinates

By Geographic Coordinates
Latitude: 0
Longitude: 0

Weather Period Settings
Hourly - next 48 hours
Daily - next 7 days

OPENWEATHERMAP DATA

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

- `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 with 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 instace that holds data about the percentage of the clouds density.
- `Unix timestamp (dt)` is long value that represents the way to track the 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 class the holds some specific system parameters like: type, ID, message (logs), country (the interrogated country), sunrise time and sunset time.

- `Timezone` represents the UTC offset represented in seconds. Example: for UTC +3:00 hours => 3 * 3600 seconds => 10800 seconds.
- `CityID` is a int value that represents the correspondent city id for the interrogated location, which can be also found on city.list.json.gz.
- `HTTPCode` is a 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 in which units will be data obtained: **Standard** (speed => meter/sec, temperature => Kelvin), **Metric** (speed =>meter/sec, temperature =>Celsius) or **Imperial** (speed => miles/hours, temperature => Fahrenheit).

OPENWEATHERONECALLAPIMAPDATA DATA

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

- `Unix timestamp (dt)` is long value that represents the way to track the time as a running total of seconds. This value is added to epoch time (1st, Jan, 1970) and constructs the current date time.
- `Geographic Coordinates` member class holds details about the geographic positioning on the globe: latitude and longitude.
- `TimezoneOffset` represents the UTC offset represented in seconds. Example: for UTC +3:00 hours => 3 * 3600 seconds => 10800 seconds.

- *Timezone* represents the name for the requested location
- *List<HourlyWeather>* is a list with *HourlyWeather* instances and 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, uvi, clouds, visibility, wind speed, wind gust, wind degree, and a list of WeatherDetails.
- *List<DailyWeather>* is a list with *dailyWeather* instances and 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, uvi, clouds, wind speed, wind gust, wind degree, and a list of WeatherDetails.
- *Units* is a Units enum value that represents in which units will be data obtained: **Standard** (speed => meter/sec, temperature => Kelvin), **Metric** (speed =>meter/sec, temperature =>Celsius) or **Imperial** (speed => miles/hours, temperature => Fahrenheit).

REDIRECTING DATA OUTSIDE OUR PLUGIN

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

Receive OpenWeatherMap data

—

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

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

Example:
_openWeatherMapModule.OnServerResponse += OnReceivingOpenWeatherMapData;
```

Receive OpenWeatherMapOneCallAPI data

—

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

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

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

Receive OpenWeatherMap exception

—

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

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

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

CONVERTING DATA FOR SIMULATION

If the subscribing from Real-Time Weather Manager to Open Weather Map Module delegate remain intact, 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 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 structure 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 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 METHOD

To request weather data from the Tomorrow, you must make 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).

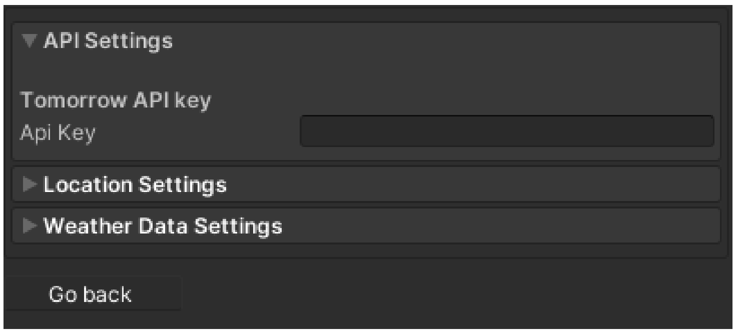
PARAMETERS

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.

INSPECTOR INTERFACE

The TomorrowModule inspector contains three main options:

- *API Settings*- are Tomorrow API settings, such as access key;
- *Location Settings*- are settings for the locality for which weather information will be requested;
- *Weather Data Settings*- are settings to define what weather information will be requested from the API.



▼ API Settings

Tomorrow API key

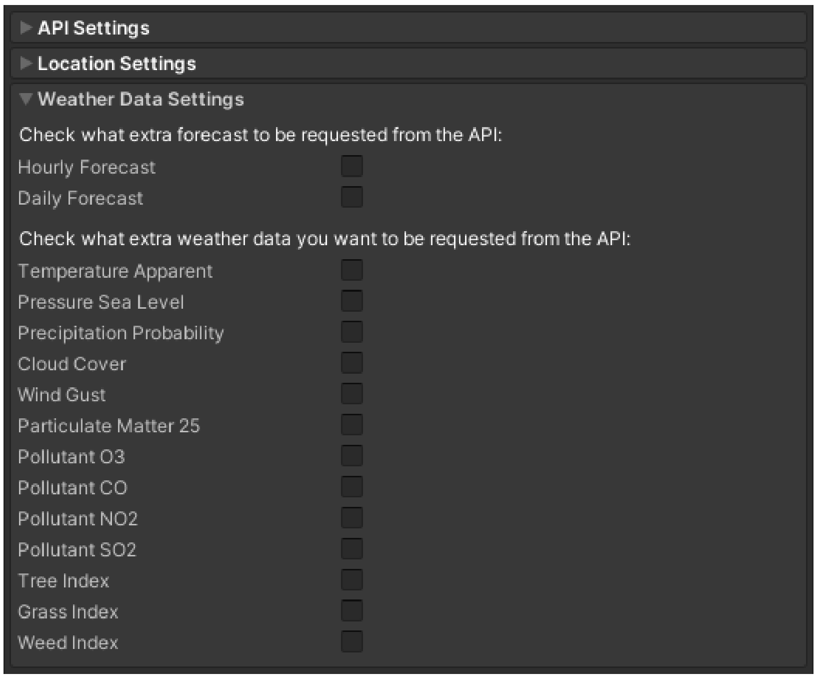
Api Key	

▶ Location Settings

▶ Weather Data Settings

Go back

Additional parameters that provide hourly/daily forecast and meteorological information related to air quality, and pollen, must be checked to be requested from the API. Otherwise, these parameters will have default values.



▶ API Settings

▶ Location Settings

▼ Weather Data Settings

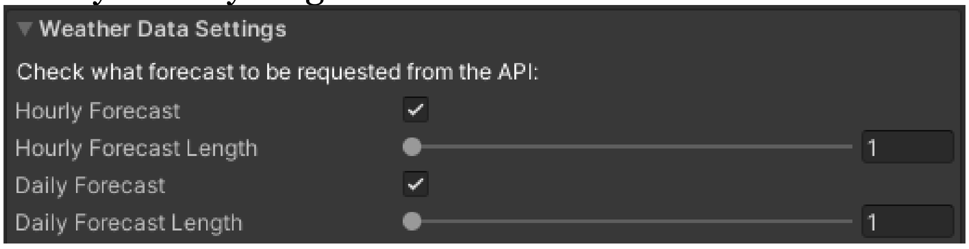
Check what extra forecast to be requested from the API:

Hourly Forecast	☐
Daily Forecast	☐

Check what extra weather data you want to be requested from the API:

Temperature Apparent	☐
Pressure Sea Level	☐
Precipitation Probability	☐
Cloud Cover	☐
Wind Gust	☐
Particulate Matter 25	☐
Pollutant O3	☐
Pollutant CO	☐
Pollutant NO2	☐
Pollutant SO2	☐
Tree Index	☐
Grass Index	☐
Weed Index	☐

Extra hourly and/or daily forecast can be set by enable corresponding checkbox. When a forecast is enabled, hourly or daily length can be set.



Weather Data Settings

Check what forecast to be requested from the API:

Hourly Forecast	✓
Hourly Forecast Length	1
Daily Forecast	✓
Daily Forecast Length	1

TOMORROW DATA

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

TomorrowData class

—

- *`latitude`* is a float value that represets 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 represents error JSON data.

TomorrowWeatherData class

—

- *`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 procentages °C;

- *`dewPoint-`* the temperature to which air must be cooled to become saturated with water vapor. Measured in procentages °C;

- *`humidity`* is a float value that represents the concentration of water vapor present in the air. Measured in procentages %;

- *`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;
- *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;
- *cloudCover-* the fraction of the sky obscured by clouds when observed from a particular location. Measured in procentages %;
- *precipitationIntensity-* the amount of precipitation that falls over time, covering the ground in a period of time. Measured in mm/hr;
- *precipitationProbability-* the chance of precipitation that at least some minimum quantity of precipitation will occur within a specified forecast period and location. Measured in procentages %.
- *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 pp;
- *pollutantNO2-* the concentration of surface Nitrogen Dioxide (NO2). Measured in ppb;
- *pollutantCO-* the concentration of surface Carbon

Monoxide (CO2). 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;
- *weatherCode*- 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
    }
```

- *precipitationType*- the PrecipitationType enum value that specify the various types of precipitation which is falling to ground level;

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 TomorrowModule class.

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

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

Receive Tomorrow exception

—

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

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

Example:
_tomorrowModule.onTomorrowExceptionRaised +=OnTomorrowExceptionRaised
```

Tomorrow API uses conventional HTTP response codes to indicate the outcome of an API request. Codes in the 2xx range indicate success, 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, the Tomorrow data must be converted to Real-Time weather data. In other words, TomorrowData class must be cast to the WeatherData class.

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

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

Example:
WeatherData rtwCurrentWeatherData =
TomorrowDataConverter.ConvertCurrentTomorrowDataToRtwData());
List<WeatherData> rtwHourlyWeatherData =
TomorrowDataConverter.ConvertHourlyTomorrowDataToRtwData());
List<WeatherData> rtwDailyWeatherData =
TomorrowDataConverter.ConvertDailyTomorrowDataToRtwData());
```

Based on timestep, *ConvertCurrentTomorrowDataToRtwData* will convert only *current* timestep data and return WeatherData, *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:
NotifyCurrentWeatherChanged(rtwWeatherData);
NotifyHourlyWeatherChanged(rtwWeatherDataList);
NotifyDailyWeatherChanged(rtwWeatherDataList);
```

REVERSE GEOCODING

ABOUT

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

The *Free Client-side Reverse Geocoding to City* API was used to implement the reverse geocoding component. This API is a free version of *Reverse Geocoding to City* API. More information about this API is available here: https://www.bigdatacloud.com/geocoding-apis/free-reverse-geocode-to-city-api.

Request method

—

To use the reverse geocoding functionality, a coroutine must be created, as in the example below, which calls 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.City);
                Debug.Log("Country=" + reverseGeoData.CountryName);
            }
}
```

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

GEOCODING DATA

The GeocodingData class is used to store and manage the geocoding information obtained from the FREE Client Side Reverse Geocoding to City API.

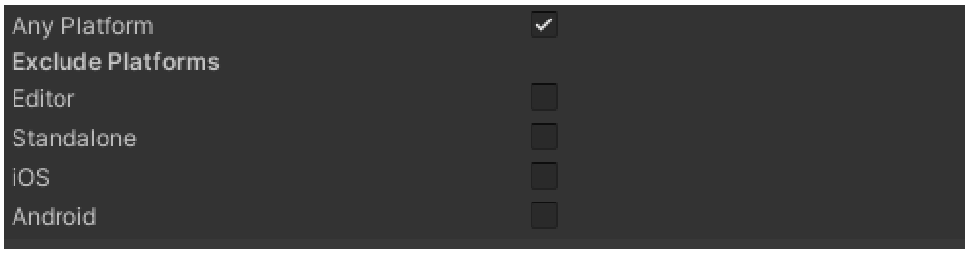
- *`latitude`* is a float value that represets a geographic coordinate that specifies the north–south position of a point on the Earth's surface. Latitude must be set according to ISO 6709.
- *`longitude`* is a float value that represents a geographic coordinate that specifies the east-west position of a point on the Earth's surface. Longitude must be set according to ISO 6709.
- *`continent`* localised Continent name in the requested language.
- *`countryName`* is a string value that represents the localised country name in the requested language.
- *`countryCode`* is a string value that represents the country code as defined by ISO 3166-1 standard.
- *`city`* is a string value that represents the localised city name in the requested language, if available.
- *`locality`* is a string value that represents the smallest geographic area recognised to which the target belongs.
- *`principalSubdivision`* is a string value that represents the localised principal subdivision name in the requested language.
- *`principalSubdivisionCode`* is a string value that represents the principal subdivision code as defined by ISO 3166-2 standard.
- *`postcode`* is a string value that represents the postcode.
- *`localityInfo`* is detailed reverse geocoded locality information

Real-Time Weather Manager DLL

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

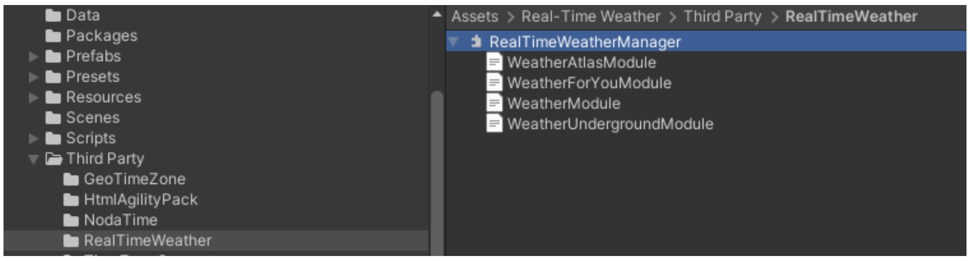
- Windows
- MacOS
- Linux/Ubuntu (UNIX)
- Android
- iOS

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



Any Platform	✓
Exclude Platforms	
Editor	☐
Standalone	☐
iOS	☐
Android	☐

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



```
Data
Packages
▶ Prefabs
▶ Presets
▶ Resources
Scenes
▶ Scripts
▼ Third Party
    GeoTimeZone
    HtmlAgilityPack
    NodaTime
    RealTimeWeather
```

Assets > Real-Time Weather > Third Party > **RealTimeWeather**

```
▼ RealTimeWeatherManager
    WeatherAtlasModule
    WeatherForYouModule
    WeatherModule
    WeatherUndergroundModule
```