



**REAL-TIME
WEATHER**

1.7.2

Documentation

Thank you for your purchase!

OVERVIEW & INSTALLATION

Installation and Quick Start	3
Enviro Integration	7
Tenkoku Integration	8
Atmos Integration	9
Expanse Integration	11
Forecast Timelapse	12
Help and Bug Report	16
Feedback Form	17
FAQ	18

SCRIPTING API

Real-Time Weather Manager	20
Weather Data	21
Atlas Module	24
Weather For You Module	25
Weather Underground Module	26
Reserve Locations	28
Open Weather Map Service	29
Tomorrow.io Service	35
Reverse Geocoding	43
Platform Compatibility	45
URP/HDRP Compatibility	46

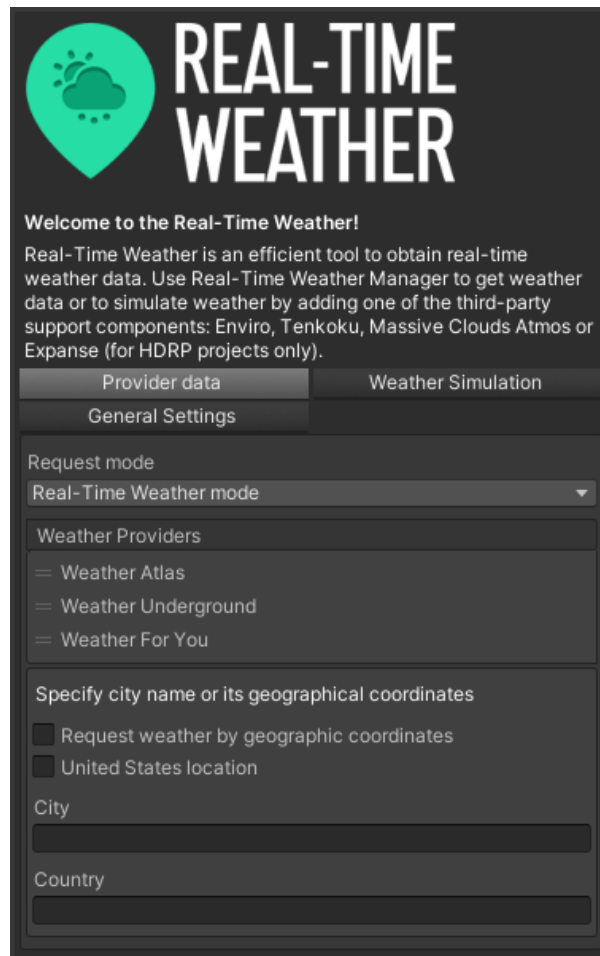
Overview & Installation

Installation

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

STEP 1

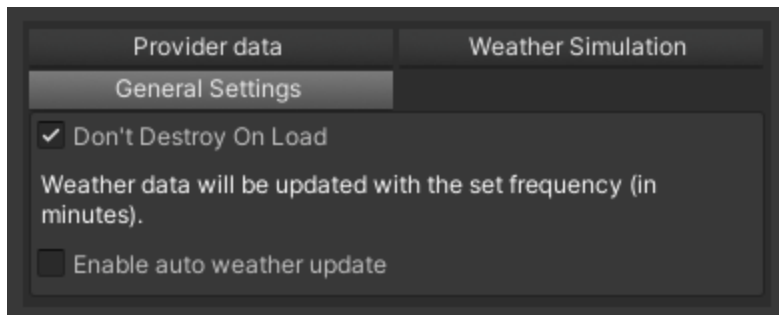
On the Unity top bar, you will find a new tab “Real-Time Weather”, so click on it and then select “Real-Time Weather Manager”. A new GameObject will be created in your scene with the “RealTimeWeatherManager” component.



STEP 2

—

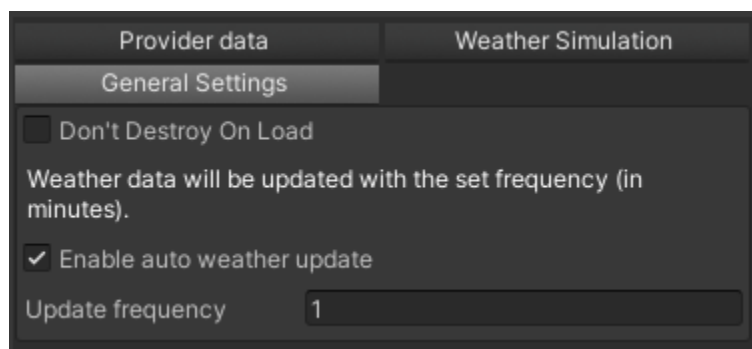
If you don't want the "Real-Time Weather Manager" to be destroyed when other scenes are loaded, check "Don't Destroy on Load" in the "General Settings" panel.



STEP 3

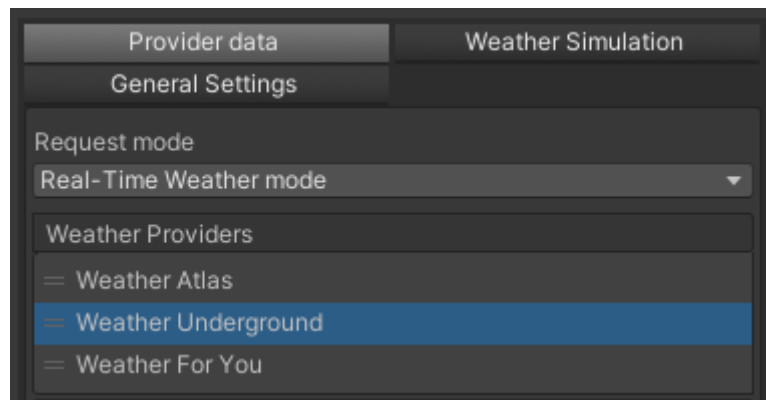
—

To get real-time weather updates, check "Enable 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 4

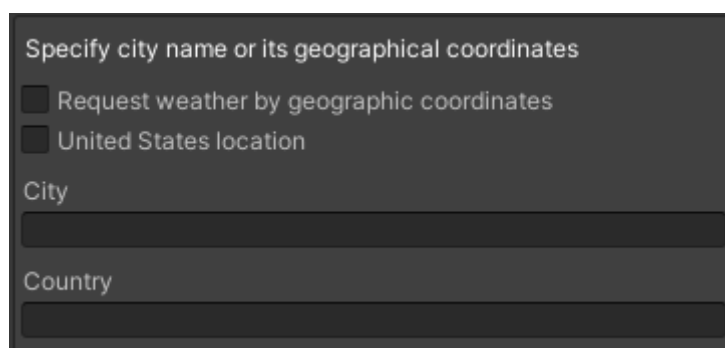
Real-Time Weather mode has 3 different weather data providers. The user can choose their priority by ordering them in the list. The first one from the top has the highest priority.



STEP 5

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.



STEP 6

Depending on what weather plugin you have in the project, you can activate either the Enviro, Tenkoku, Atmos or Expanse simulation. Simply click on the corresponding button, and everything that the weather simulation needs will be added to your scene. If there already is a valid simulation detected in the scene, it will try to use that instead of creating a new one.



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

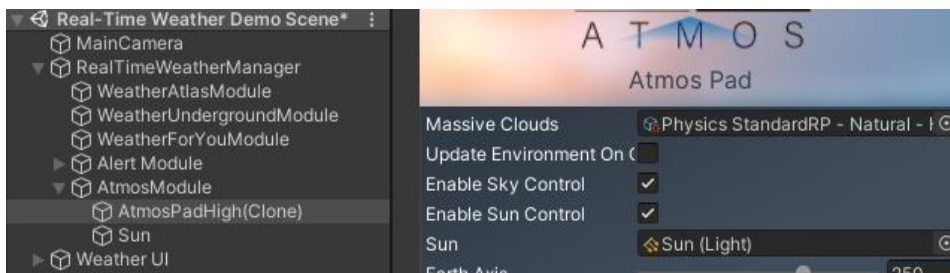
Launch the Setup Wizard from the menu: Window -> Massive Clouds Atmos -> Setup Wizard. From the Project Settings, press the Fix Now button on the Preloaded Shaders.

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

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.

Forecast Timelapse

About

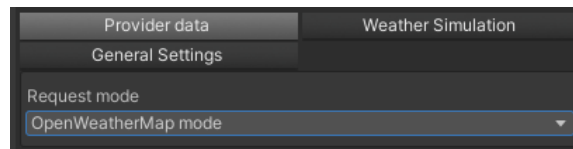
"Forecast Timelapse" is a feature that allows the user to create a simulation of the atmospheric conditions for several hours or days into the future. This works by fetching weather data from one of the supported providers and creating a speed-up simulation based on that data. The simulation length can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

ACTIVATION

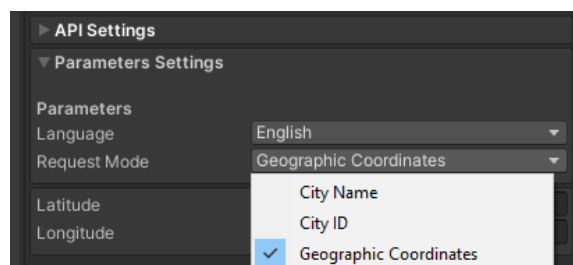
So far, only two weather providers support this mode and those are OpenWeatherMap and Tomorrow.io. We will go into detail on how to set up the forecast mode for each of these providers.

Working with OpenWeatherMap

To setup forecasting mode for OpenWeatherMap, first we need to select the provider from the "Request Mode" dropdown list in the "General Settings" section.

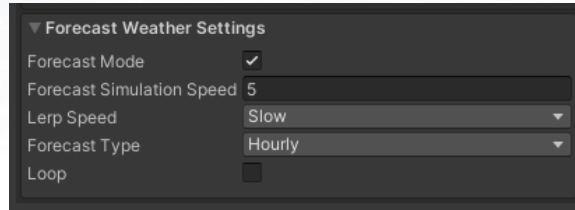


After setting up the "API Key" parameter, we need to select the "Geographic Coordinates" option from the "Parameters Settings" -> "Request Mode" dropdown list. This will enable us to input the latitude and longitude values of the location that we will use for our weather simulation.



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 adjusting the forecast simulation.

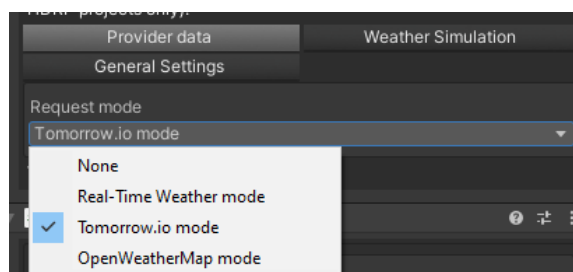


From here we can set a few simulation parameters:

- 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.
- Forecast Type: options -> "Hourly", "Daily", represents the simulation type and the amount of data fetched from the provider. "Hourly" will provide data for the next 48 hours while "Daily" will provide data for the next 7 days.
- 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.

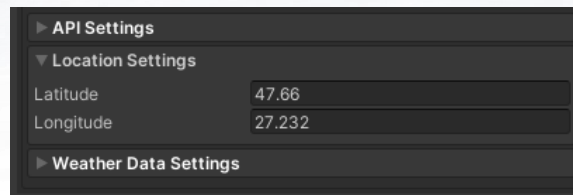
Working with Tomorrow.io

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.

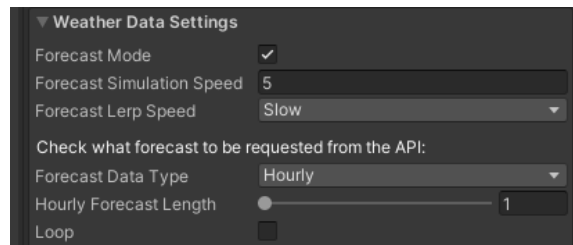


Forecast Timelapse

After setting up the "API Key" parameter we need to input the latitude and longitude parameters in the "Location Settings" section.



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



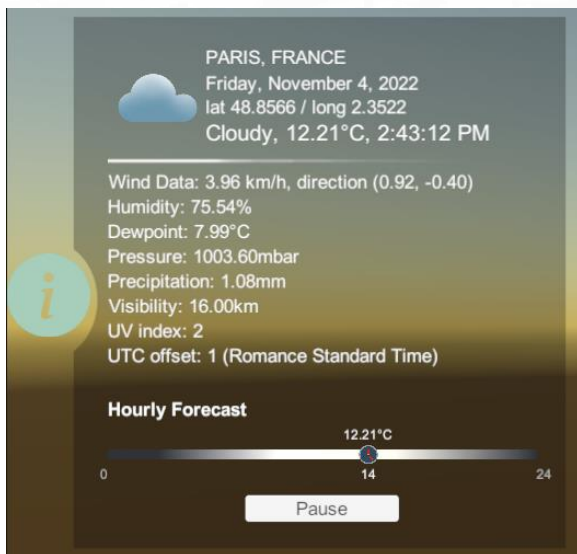
From here we can set a few simulation parameters:

- 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.
- Forecast Type: options -> "Hourly", "Daily"; represents the simulation type and the amount of data fetched from the provider.
 - Hourly Forecast Length: specifies the number of hours that will be simulated, up to a max of 108 hours;
 - Daily Forecast Length: specifies the number of days that will be simulated, up to a max of 14;
- 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:

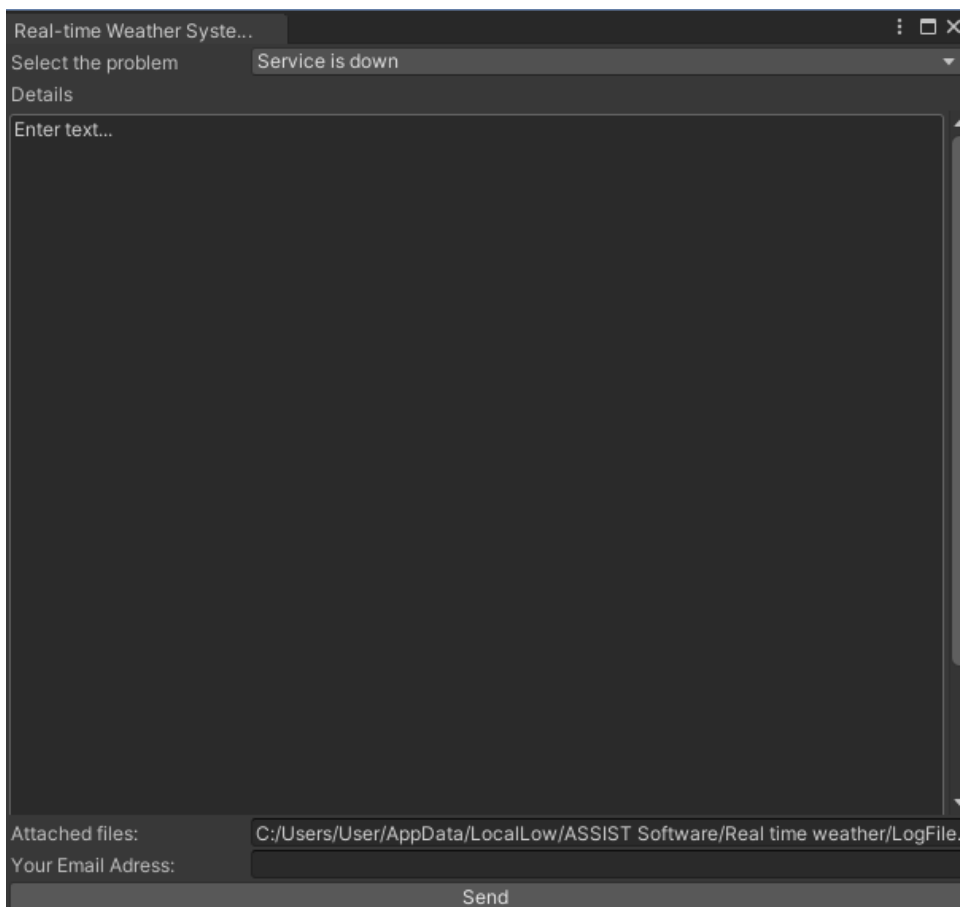
```
ForecastModule.instance.OnForecastProgressModuleTick += OnForecastUpdate;
```


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 fails to obtain weather. It can also be activated by going to the “Real-time Weather Manager” tab and selecting “Help”, then “Bug Report”.



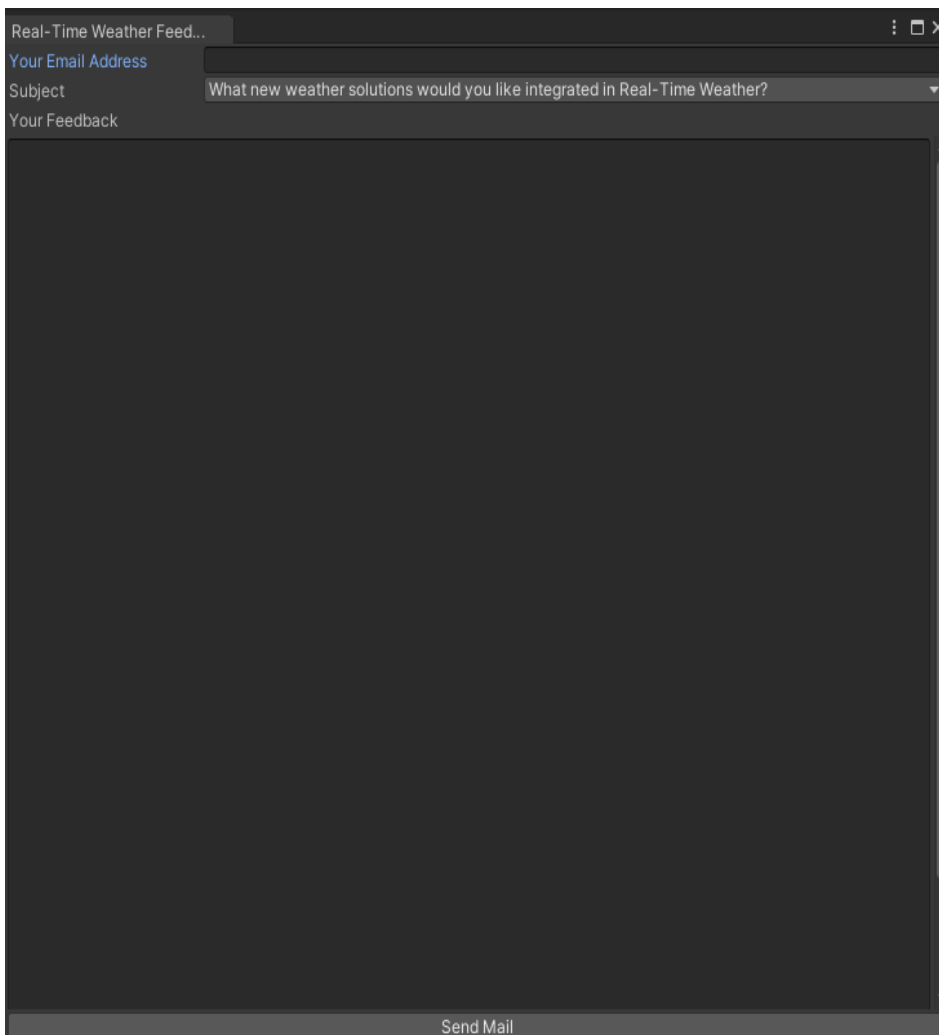
The screenshot shows a dialog box titled "Real-time Weather System...". It features a dropdown menu for "Select the problem" with "Service is down" selected. Below this is a "Details" section with a large text area labeled "Enter text...". At the bottom, there is a field for "Attached files:" containing the path "C:/Users/User/AppData/LocalLow/ASSIST Software/Real time weather/LogFile...", a field for "Your Email Adress:" (note the typo), and 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 dialog box titled "Real-Time Weather Feed...". It contains the following fields:

- Your Email Address**: A text input field.
- Subject**: A dropdown menu with the selected text "What new weather solutions would you like integrated in Real-Time Weather?".
- Your Feedback**: A large text area for entering feedback.
- Send Mail**: A button at the bottom right of the dialog.

FAQ

Q: I don't know how to use the Real-Time Weather plugin, do you have a tutorial?

A: Yes, you can check our tutorials from the Unity Asset Store product page. You could also read the **Overview & Installation** chapter from the documentation, which has enough information 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 you use. For example, the Tenkoku asset works only on SRP, the Expanse asset works only on HDRP.

Q: Is this asset providing forecast simulations?

A: Yes, you can simulate complete forecast scenarios (using weather data from Tomorrow.io or OpenWeatherMap), but only for weather data.

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

A: Ensure that you made 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 use Real-Time Weather mode, but I don't receive any weather data.

A: If the weather data for some specific locations are not returned from weather providers, then change the request to a nearby location. Also, we are pleased 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 for number of requests?

A: You have 3 providers for weather data, the first one (Real-Time Weather mode), that has no limit, and the API providers (Tomorrow.io and OpenWeatherMap), which are constrained to different limitations based on the subscription.

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 extends to the high distances, and is more visible when the ground is positioned above the pivot of the clouds object. The developers team suggests that in that 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 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, 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 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 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.



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:

```
_weatherForYouModule.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:

```
_weatherForYouModule.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`.

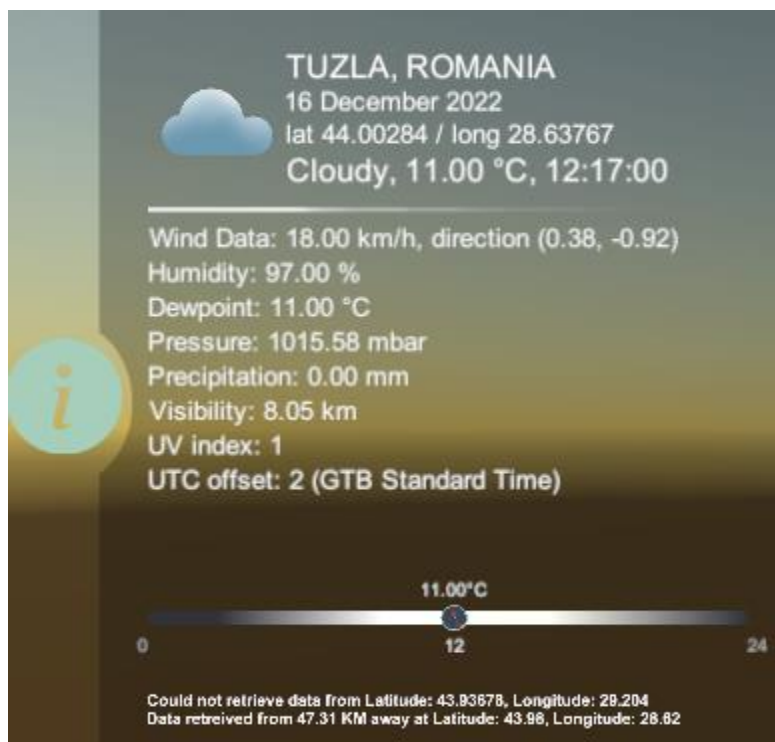
Reserve Locations (only for Geographic Coordinates input)

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 [\[WeatherStations\]](#).

Will be provided up to five of the closest weather stations, replacements for the user-given coordinates. If all the replacements 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



OpenWeather

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 more [[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, or the next 7 days with a daily forecast.

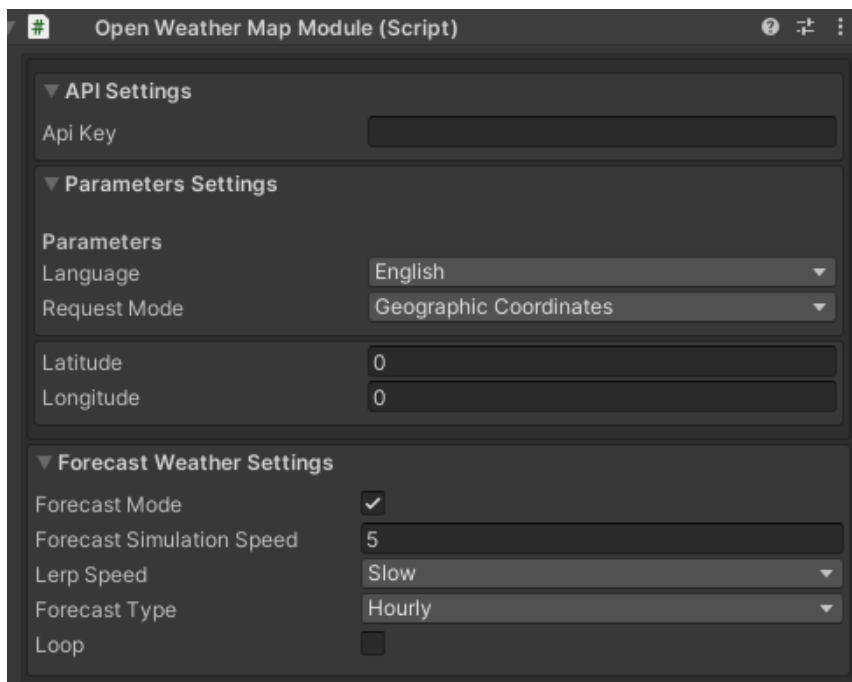
The following link is used:

```
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 **excluded**, 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



Openweather map data

The OpenWeatherMap Data class is used to store and manage the weather data from obtained by the 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 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 long value that represents the way to track time as a running total of seconds. This value is added to epoch time Jan, 1970) and constructs the current date time.
- *System Data* is a class that holds some specific system parameters like / 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.
- *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` class is used to store and manage the weather data obtained from the One Call API service:

- *Unix timestamp (dt)* is long value that represents the 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.
- *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 be also 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 EXCEPTION

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 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 data from 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

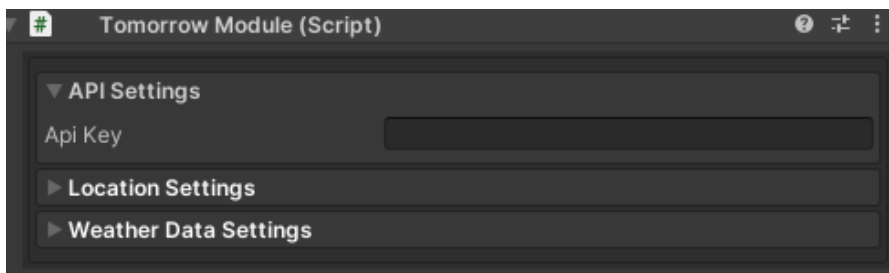
—

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.

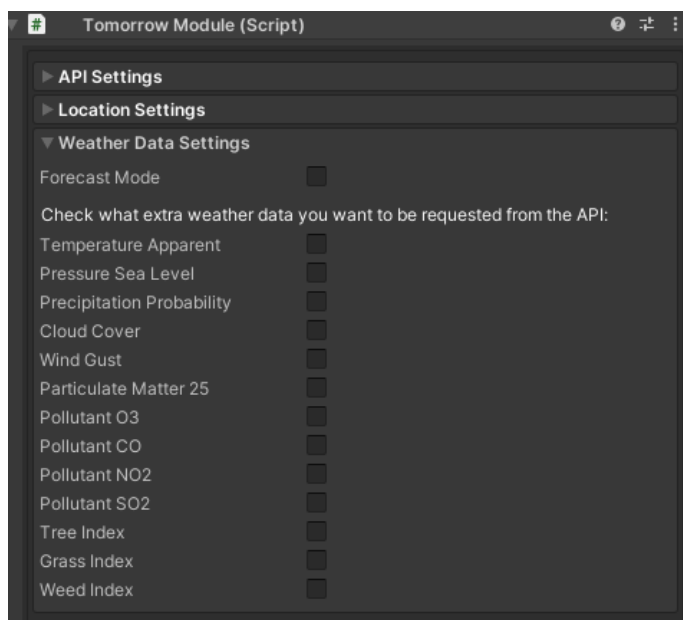
Inspector interface

The TomorrowModule inspector contains three main options:

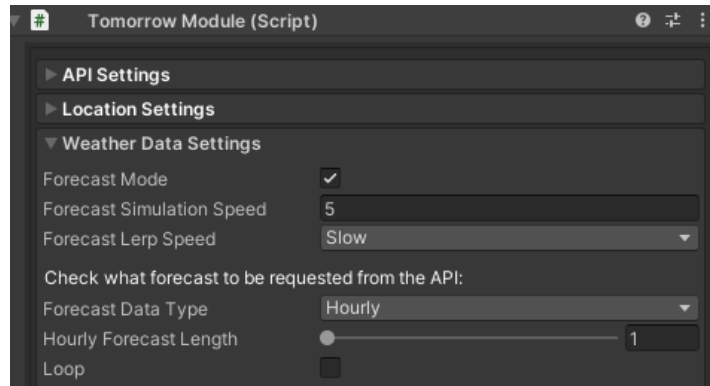
- *API Settings* – are Tomorrow API settings, such as access key;
- *Location Settings* – settings for the location from where the weather information will be requested;
- *Weather Data Settings* – settings that define what weather parameters will be requested from the API.



Additional parameters that provide 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.



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



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 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 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 percentages °C;
- *dewPoint* - the temperature to which air must be cooled to become saturated with water vapor. Measured in percentages °C;
- *humidity* is a float value that represents the concentration of water vapor present in the air. Measured in percentages %;
- *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 percentages %;
- *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 percentages %.
- *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 specifies 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 onTomorrowDataSent;
```

Example:

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

RECEIVE TOMORROW EXCEPTION

Receiving Tomorrow exceptions can be done by subscribing to then `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 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:
 - *neighbourhood* 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 localised municipality (not necessarily a city).
 - *village/town/city* is a string value that represents the localised village/town/city name.
 - *county/district* is a string value that represents the localised county/district name (could be shown if no locality is found).
 - *state* is a string value that represents the localised state name.
 - *country* is a string value that represents the localised country name.
 - *countryCode* is a string value that represents the country code as defined by ISO 3166-1 standard.

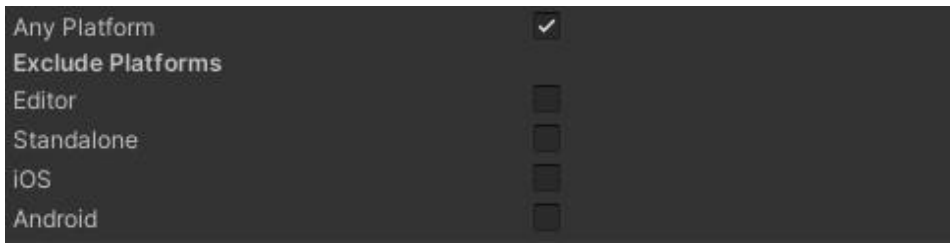
Platform Compatibility

Real-Time Weather Manager DLL

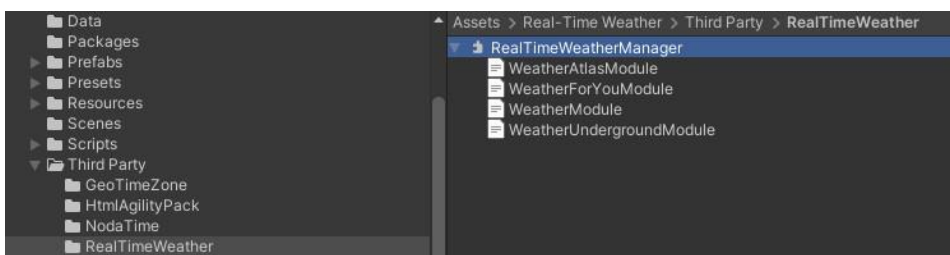
Real-Time Weather is a stable solution for the following platforms:

- 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!