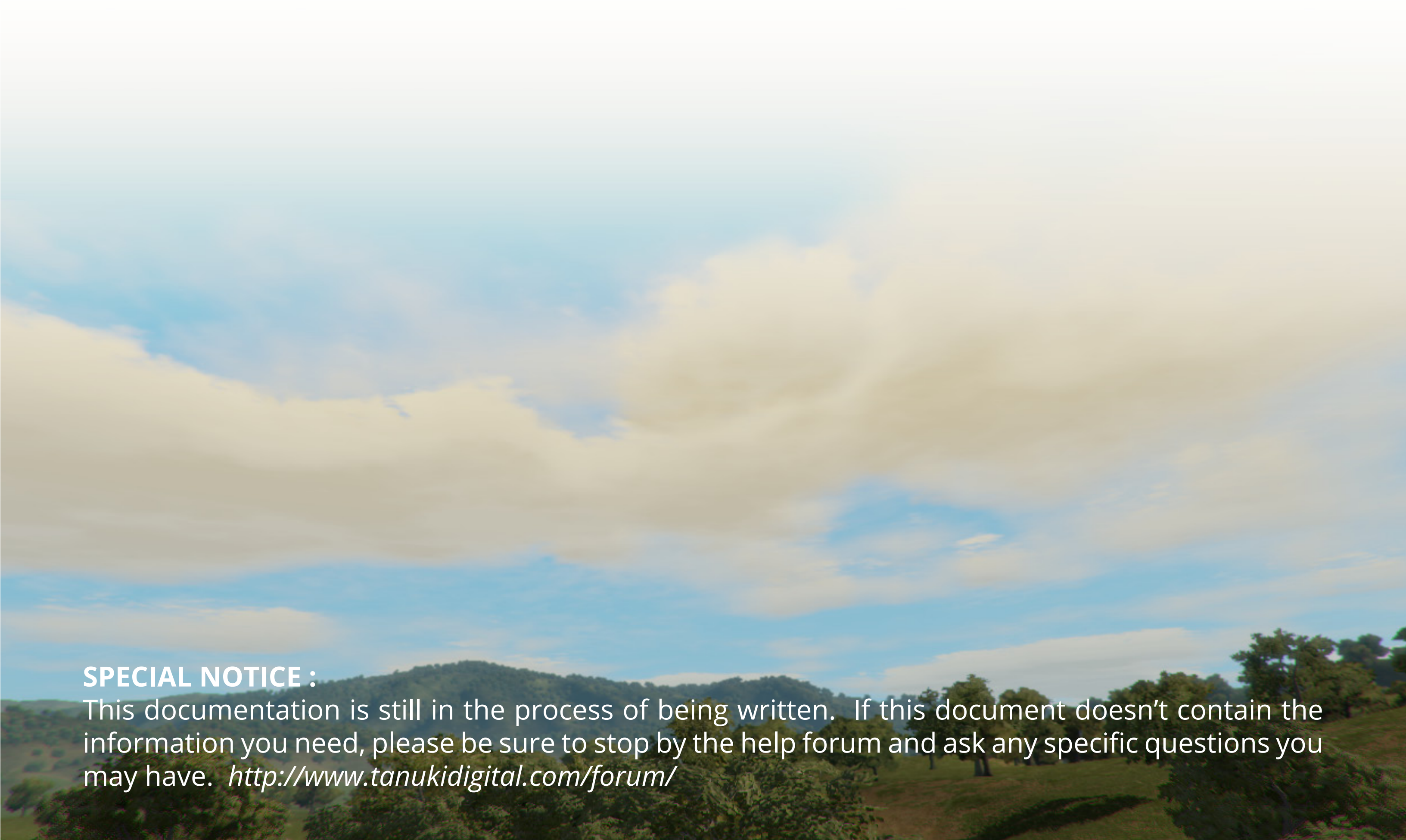
TENKOKU

DYNAMIC SKY SYSTEM

OFFICIAL DOCUMENTATION

Based on Tenkoku version 1.1.1

Documentation last updated on 7/5/2016



SPECIAL NOTICE :
This documentation is still in the process of being written. If this document doesn't contain the information you need, please be sure to stop by the help forum and ask any specific questions you may have. *http://www.tanukidigital.com/forum/*

TABLE OF CONTENTS

GETTING HELP & CONTACT INFO

This documentation has been written with the goal of giving Tenkoku users an in-depth overview of the various components of the system. There are various other resources online in which you can find more information about using Tenkoku. if you have specific questions, please head over to the Tanuki Digital users forum and browse the posts, or make your own:

http://tanukidigital.com/forum/

and of course if for any reason you would like to ask a question directly, please feel free to email us here:
konnichiwa@tanukidigital.com

Tenkoku Homepage:
http://tanukidigital.com/tenkoku/

Tenkoku Demo Scene:
http://tanukidigital.com/tenkoku/demo

This Documentation file:
http://tanukidigital.com/tenkoku/documentation

INSTALLATION

STEP 1. IMPORT TENKOKU FILES INTO YOUR PROJECT

Go to: "Assets -> Import Package -> Custom Package..." in the Unity Menu and select the "tenkoku_dynamicsky_ver1.x.unitypackage" file. This will open an import dialog box. Click the import button and all the Tenkoku files will be imported into your project list.

STEP 2. ADD THE TENKOKU MODULE TO YOUR SCENE

1) Drag the **Tenkoku DynamicSky** prefab located in the "/PREFABS" folder into your scene list.
2) If it isn't set already, make sure to set the Tenkoku DynamicSky's position in the transform settings to 0,0,0

STEP 3. ADD TENKOKU FX TO YOUR CAMERA

1) Click on your main camera object and add the Tenkoku Fog effect by going to the top menu under **"Component-->Image Effects-->Tenkoku-->Tenkoku Fog"**. For best results this effect should be placed **before** most of your other camera image effects, including Tonemapping (if applicable).

(optional)
2) Click on your main camera object and add the Tenkoku Sun Shaft effect by going to the top menu under **"Component-->Image Effects-->Tenkoku-->Tenkoku Sun Shafts"**. For best results this effect should be placed **after** your Tonemapping effect (if applicable).

A NOTE ABOUT SCENE CAMERAS:

Tenkoku relies on tracking your main scene camera in order to properly update in the scene. By default Tenkoku attempts to auto locate your camera by selecting the camera in your scene with the 'MainCamera' tag. Alternatively you can set it to manual mode and drag the appropriate camera into the 'Scene Camera' slot.

A NOTE ABOUT LIGHTING:

Tenkoku includes it's own built-in lighting for your scene, so in most cases having additional directional lights in your hierarchy that light your scene objects is not only unnecessary, but could also possibly interfere with the way Tenkoku lights the scene. Because of this it's recommended to not have additional directional lights in your scene.

A NOTE ABOUT UPGRADING:

When upgrading between different versions of Tenkoku, it's best to completely uninstall the current version first by deleting the "Tenkoku DynamicSky" object from your scene, deleting the Tenkoku image effects from your camera, and also deleting the Tenkoku folder from your project list, before installing an updated version.

GENERAL NOTES

A NOTE ABOUT C#:

As of Tenkoku Version 1.1.1 the codebase has been rewritten in C#. Tenkoku components are now contained under the **Tenkoku.Core** namespace. Make sure to update your code accordingly!

A NOTE ABOUT COLOR: GAMMA vs LINEAR

Tenkoku has been designed with Linear color workflows in mind by default. While gamma color settings are supported, the color rendition of Tenkoku is best when used in conjunction with Linear mode, HDR camera settings, and camera Tonemapping image effects.

Multiple Image effects on a camera can also look better or worse, depending on the order in which they are stacked. The following order is recommended for best visual fidelity... Bloom-->Tonemapping-->SunShafts --> Antialiasing.

A NOTE ABOUT ACCURACY:

Moon and planet calculations are currently accurate for years ranging between 1900ca - 2100ca. The further away from the year 2000ca that you get (in either direction) the more noticeable calculation errors will become. Additional calculation methods are currently being looked at to increase the accuracy range for these objects.

FUTURE IMPROVEMENTS

Below are some additional features and improvements coming in future Tenkoku updates. If you have any bugs, issues, or features you'd like to see in Tenkoku then please lets us know either via email or the forums!

Improved weather effects
We're currently looking at the addition of other effects such as sandstorms, rain lens effects, and improved fog.

Advanced Weather Automation
Calculation of weather based on season, latitude, and temperature shift.

Cloud Shadow Casting
scene-wide shadows cast from Cumulus cloud objects.

SCENE CUSTOMIZATION

SET ORIENTATION

Tenkoku automatically aligns the sky sytem to your camera, pointing at a default orientation where **Z = North** and **X = East.** If you need to change the orientation so that north and east are in different positions relative to your scene, then you can change the **Set Orientation** setting under the **Configuration** tab.

SET LATITUDE and LONGITUDE

The position of the sky objects are determined by the position of the "earth" from which it is viewed. You can change this position by setting latitude and longitude under the **Time and Position** tab. Note that regardless of the longitude setting, the time and date settings will always be listed at the default **GMT** setting, so currently you'll need to account for the time-zone difference when changing longitude.

TIME COMPRESSION

You can speed up or slow down the passage of time by using the AdvanceTime setting. When enabled, the AdvanceTime setting will advance time according to the set compression rate (default x20). By default the time compression is constant across all hours of the day, however you can edit the Speed Curve setting to change the variability of the compression over the course of the day (in order to speed up time at night for example). In the speed X plane 0 = 12am, and 1 = 11:59pm. the speed Y plane is a time multiplier, so 1 = the current time compression, where 2 = twice the current time compression, and so forth.

SETTING CUSTOM COLORS

Under the **Configuration** tab there are a few ways you can customize the color rendering of Tenkoku to suit your scene. You can quickly change the overall sky tint, or ambient shadow tint, or you can edit the color lookup texture to more finely control how colors occur over time.

Overall Tint - This applies an overall color tint to your entire sky and lighting system, regardless of time of day. The strength of the color overlay can be controlled by changing the color alpha value.

Sky Tint - This applies a color tint only to the daylight sky rendering. The strength of the color overlay can be controlled by changing the color alpha value.

Ambient Tint - This applies an overall color tint to the shadow areas used in your scene. The strength of the tint can be controlled by changing the color alpha value.

Skybox Ground - This adjusts the ground color of the background skybox and can influence ambient light calculations. By default this is set to black.

Color Texture - Tenkoku uses this texture as a color lookup table for rendering various objects over time. You can edit this texture as you need to best fit your scene. The width of the texture is broken down into a 12-hour cycle, where the left side represents morning starting at 12am, the middle represents 6am around when the sun is generally heading into the Dawn/Dusk position, and the right side represents 12 noon when the sun is at it's peak. The color lookup proceeds back to the left in the afternoon/evening from 12 noon to midnight.

SETTING CUSTOM NIGHT SKY

By default, Tenkoku calculates and renders it's own accurate night sky and star positions, however if you would rather use a custom skybox texture for the night sky you can do so under the **Celestial Settings** tab by first turning the **Star Rendering** setting to "Off", turn the **Galaxy Rendering** setting to "On", and then adding your own custom sphere-mapped image on the **Galaxy Texture** slot.

Note: This needs to be a Texture 2D sphere-mapped image, not a cubemap! (see the included example texture)

SCENE CUSTOMIZATION

CUSTOMIZING WEATHER

There are currently 3 weather mdoes to choose from in Tenkoku....

Manual - This setting allows you to manually set tolerances for each weather attribute, from clouds to precipitation and fog effects. The "Link Clouds to Timer" checkbox will advance the clouds based on the autotimer clock settings. So if you have timer settings to advance at 100X normal speed, you'll see the clouds moving at an extremely fast rate. To move clouds at a more realistic rate regardless of the autotimer settings then keep this disabled.

Automatic (Random) - This setting automatically generates random weather patterns according to the set timeframes. The **Weather Pattern Lifetime** variable designates (in minutes) how long each weather pattern will last for before choosing and transitioning to the next pattern. The **Weather Transition Speed** variable designates how long (in minutes) a transition takes between pattern A and pattern B.

Automatic (Advanced) - Coming Soon!

SUIMONO 2.0 INTEGRATION

NOTE: Currently, getting sky information into Suimono requires that Dynamic Reflections be turned on in the Suimono Module and Surface objects.

Also note that in the future Suimono will be able to cross-reference wind and weather information as well, providing automatic wave flow and magnitude adjustments for water surfaces based on tenkoku wind/weather settings

Below are some specific notes to keep in mind when using Suimono and Tenkoku together.

SET REFLECTIONS TO RENDER SKY

If the Tenkoku sky is not reflecting in your Suimono water surface then you likely need to tell the Suimono reflections to render the correct game layers. First, go to the **SUIMONO_Surface** --> **reflectionObject** game object and add the **"Sky"** and **"Moon"** layers to the **"Reflect Layers"** setting.

SET CAUSTICS LIGHTING

If you're using the built-in Suimono caustic funtion, then you'll need to make a few adjustments on how the caustics behave in order to work best with Tenkoku. First, go to the **SUIMONO_Module** --> **caustic_effects** game object and remove the Tenkoku "sky" and "moon" layers from the **Use these layers** setting.

Also you'll want to drag the **Tenkoku DynamicSky** --> **SkySphere** --> **LIGHT_World** game object onto the Tenkoku **Scene Light Object** slot. This should adjust caustic strength based on scene lighting and prevent the caustic lights from being enabled at night time.

ENABLE TENKOKU INTEGRATION

As of Suimono version 2.1 there is now a "Tenkoku Sky System - Integration" section on the Suimono_Module object when both Suimono and Tenkoku are available in the same scene. From here you can enable cross-referencing of Tenkoku wind direction and strenght settings from Tenkoku to Suimono. This will automatically override Suimono direction and wind settings.

KNOWN-ISSUES

There are a few integration issues between Tenkoku and Suimono that are still being worked out...

1) Tenkoku aurora effects do not render in Suimono reflections.

2) Tenkoku atmospheric fog is not overlaying Suimono water surfaces correctly in deferred rendering. A future Suimono update will address this issue.

3) As noted above weather and wind direction links between both systems will be integrated in a future Suimono update.

Programming and Functions

A number of Suimono functions can be accessed via code in order to perform advanced functions outside of the built-in Suimono functionality. The below guide will give you a number of examples of how to access raw system data for use in your coding projects.

INTRODUCTION I. Referencing the Tenkoku Module object

Most of the functions listed below first require you to locate the main Tenkoku object that is located in your scene. It's best to locate this object once and store a reference to it that you can continually access later. In the below example code, we make a global variable reference, and then do a search to locate the module in the Start() function. Note that in C# (Tenkoku Version 1.1.1 and higher) you need to locate the Tenkoku Module under the Tenkoku.Core namespace.

```
Tenkoku.Core. TenkokuModule tenkokuModule;

void Start(){
	tenkokuModule = GameObject.Find("Tenkoku DynamicSky").gameObject.GetComponent<Tenkoku.Core.TenkokuModule>();
}
```

Once you've stored this reference to the Tenkoku Module object, you can then use it later in your code to access built-in functions and variables. The code examples below assume that you've already made this reference elsewhere in the same script file.

TENKOKU TIME AND POSITION. Accessing Individual Tenkoku Time variables

Once you've referenced the main module as detailed above, you can then directly access and change different Tenkoku variables in your scene. Below is a list of the specific time and position variables that are available. Please note that time variables are always set as GMT0 time and do not take into account local longitudinal shift.

```
tenkokuModule.currentYear = 2015;
tenkokuModule.currentMonth = 3;
tenkokuModule.currentDay = 21;
tenkokuModule.currentHour = 7;
tenkokuModule.currentMinute = 30;
tenkokuModule.currentSecond = 0;

tenkokuModule.setLatitude = 40.2f;
tenkokuModule.setLongitude = 0.0f;
```

currentYear (int) - gets or sets the current year in the Gregorian calendar. Years in BCE are listed as negatives.
currentMonth (int) - gets or sets the current month, range 1-12.
currentDay (int) - gets or sets the current day of the month, ranges 1-30 or 1-31 depending on month.
currentHour (int) - gets or sets current hour, range 0-23.
currentMinute (int) - gets or sets current minute, range 0-59
currentSecond (int) - gets or sets current second, range 0-59

setLatitude (float) - gets or sets global latitude position. Ranges between -90.0 and 0.0 are in the southern hemisphere, and ranges between 0.0 and 90.0 are in the northern hemisphere. 0.0 is equal to the equator.

setLongitude (float) - gets or sets global longitude position. Ranges between -180.0 and 180.0, where 0.0 = GMT0. Unless you absolutely need longitude positioning in your project I recommend keeping this set at 0.0, just for the sake of simplicty.

TENKOKU WEATHER I. Accessing Individual Weather variables

Once you've referenced the main module as detailed above, you can then directly access and change different Tenkoku variables in your scene. Below is a list of the specific weather related variables that are available.

```
tenkokuModule.weather_cloudAltoStratusAmt = 0.1f;
tenkokuModule.weather_cloudCirrusAmt = 0.2f;
tenkokuModule.weather_cloudCumulusAmt = 0.5f;
tenkokuModule.weather_OvercastAmt = 0.0f;
tenkokuModule.weather_cloudScale = 1.0f;
tenkokuModule.weather_cloudSpeed = 0.2f;
tenkokuModule.weather_RainAmt = 0.0f;
tenkokuModule.weather_SnowAmt = 0.0f;
tenkokuModule.weather_FogAmt = 0.0f;
tenkokuModule.weather_FogHeight = 500.0f;
tenkokuModule.weather_WindAmt = 0.3f;
tenkokuModule.weather_WindDir = 180.0f;
tenkokuModule.weather_temperature = 180.0f;
tenkokuModule.weather_rainbow = 180.0f;
tenkokuModule.weather_lightning = 0.0f;
tenkokuModule.weather_lighningDir = 60.0f;
tenkokuModule.weather_lighningRange = 180.0f;
```

weather_cloudAltoStratusAmt (float) [0.0-1.0] - Controls the amount of mount of Alto Stratus Clouds.
weather_cloudCirrusAmt (float) [0.0-1.0] - Controls the amount of mount of Cirrus Clouds.
weather_cloudCumulusAmt (float) [0.0-1.0] - Controls the amount of mount of Cumulus Clouds.
weather_OvercastAmt (float) [0.0-1.0] - Controls the amount of overcast clouds.
weather_cloudScale (float) - Controls the frequency scale of all cloud layers, making clouds appear bigger or smaller.
weather_cloudSpeed (float) - Controls how fast clouds move in reaction to the below wind variables.

weather_RainAmt (float) [0.0-1.0] - Amount of rainfall.
weather_SnowAmt (float) [0.0-1.0] - Amount of snowfall.
weather_FogAmt (float) [0.0-1.0] - Amount of ground fog.
weather_FogHeight (float) - Maximum height of fog (in world-space).

weather_WindAmt (float) [0.0-1.0] - Amount of wind.
weather_WindDir (float) [0.0-359.0] - Direction of wind movement (in degrees).

weather_temperature (float) - current temperature in fahrenheit.
weather_rainbow (float) [0.0-1.0] - visibility of Rainbow.

weather_lightning (float) [0.0-1.0] - frequency in which lighning strikes will render.
weather_lightningDir (float) [0.0-360.0] - Direction (relative to camera) of lightning strikes.
weather_lightningRange (float) [0.0-360.0] - A randomized arc (relative to lightningDir) where lightning will be generated.

TENKOKU WEATHER II. Accessing Random Weather Generator

Tenkoku currently has an automatic(random) weather generator that you can use to automatically modulate weather patterns over time. The code below will turn on the auto weather, set the weather pattern to play for 5 minutes, and set the between-pattern transition to about 10 seconds. However it's best to also set a boolean variable (below called randomizePattern) elsewhere in your code so that you can force a new pattern update when you want to.

```
tenkokuModule.weatherTypeIndex = 1;
tenkokuModule.weather_autoForecastTime = 5.0f;
tenkokuModule.weather_TransitionTime = 0.1f;
if (randomizePattern){
    randomizePattern = false;
    tenkokuModule.weather_forceUpdate = true;
}
```

weatherTypeIndex (int)
0 = manual mode. You'll need to set all weather variables manually, either in the editor or via code.
1 = Automatic Mode(Random). Sets weather from randomly generated variables.
2 = Automatic Mode (Advanced). Currently turned off... coming soon!

weather_autoForecastTime (float) - time in seconds that a weather pattern will play for.
weather_TransitionTime (float) - Time in seconds that it takes to transition from one weather pattern to another.
weather_forceUpdate (bool) - force the random weather generator to generate and start next weather transition.

TRACKING CAMERA. Accessing the Tenkoku Tracking Camera

In some cases it's desirable to access and change the Tenkoku tracking camera variable while your scene is playing, instead of just setting it at the beginning from the inspector. After accessing the Tenkoku Module above you can directly access and change the camera that Tenkoku uses to track around the scene.

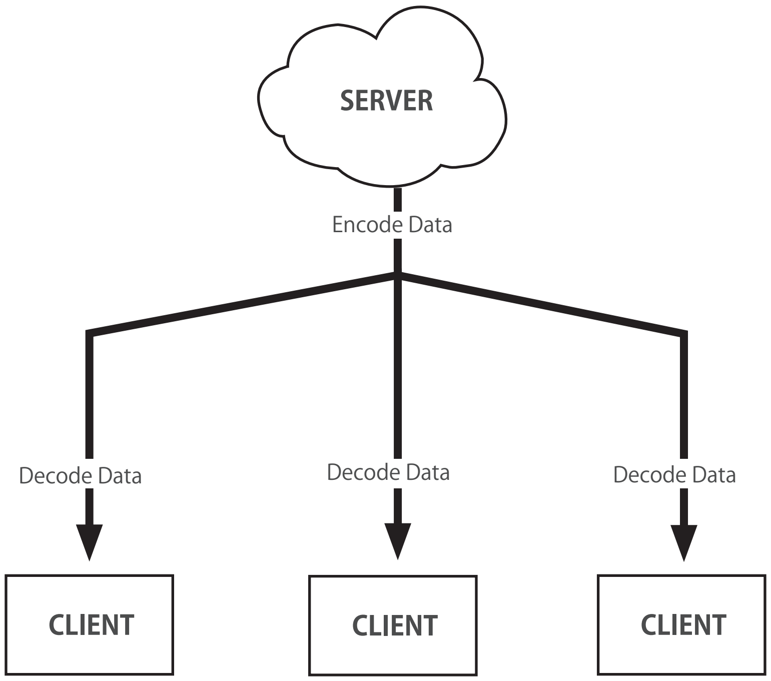
```
Transform useCamera;

void LateUpdate () {
    if (tenkokuModule != null){
        if (useCamera != null){
            tenkokuModule.cameraTypeIndex = 1;
            tenkokuModule.manualCamera = useCamera;
        }
    }
}
```

manualCamera (Transform) - Get or set the current camera tracked by the Tenkoku system.

cameraTypeIndex (int) - set the type of tracking mode. 1 = manual, and 0 = auto. When set to autoTenkoku will track the current camera in your scene designated with the 'MainCamera' tag.

SYNCING NETWORK DATA. Sharing Tenkoku Variables over a server.

Tenkoku version 1.0.4 adds a couple new functions that are useful for easily sharing most Tenkoku variables over a server/client environment. This can be useful for syncing tenkoku data between players by having each client inherit the master server settings.



function Tenkoku_EncodeData() : String;

This function automatically encodes Tenkoku system data into a single given String variable, which can then be shared over the network using your network manager of choice. Currently this functions stores date, time, latitude, longitude, and all cloud and weather settings. This should be encoded from the Server Tenkoku Settings and then passed to each client machine after it's encoded.

```
string saveData;
saveData = tenkokuModule.Tenkoku_EncodeData();
```

function Tenkoku_DecodeData(dataString : String);

This function automatically decodes data fromt he given encoded String, and sets the Tenkoku system to use all the encoded settings. Currently this functions decodes date, time, latitude, longitude, and all cloud and weather settings. Note that the example below assumes that you've already passed encoded data into the "saveData" String.

```
string saveData;
saveData = [get this data from the server]
tenkokuModule.Tenkoku_DecodeData(saveData);
```

USING C#. Sharing data between Javascript and C#. (DEPRECATED)

NOTE: AS of Suimono Version 2.1.2 all components have been rewritten in C#! The workaround detailed below is specific for older versions of Tenkoku written in .JS, and is now deprecated. It is recommended to work directly in C# to access Tenkoku components, variables and function. Please update your code accordingly.

The current version of Tenkoku is written in Javascript. While there are plans on the horizon for doing a complete C# conversion of Tenkoku, there is a new helper function added to Tenkoku version 1.0.4 that should allow much easier communication between your C# scripts and the Tenkoku system.

function Tenkoku_SetData(data : String); (javascript)
SendMessage("Tenkoku_SetData", "function(data)"); (C#)

This function allows you to set specific variables in Tenkoku (such as time, date, and weather settings) without having to directly communicate from one script to another. Instead its recommended to use Unity's SendMessage function as illustrated in the C# script below.

```
private GameObject tenkokuModule;

void Start () {
   tenkokuModule = GameObject.Find("Tenkoku DynamicSky");
}

void Update (){
   tenkokuModule.SendMessage("Tenkoku_SetData","currentHour(8)",SendMessageOptions.DontRequireReceiver);
}
```

The above example sets Tenkoku's hour to 8 in the morning, by first calling the main function "Tenkoku_SetData" and then calling a sub function "currenthour" and passing a variable "8". The sub-functions are labelled as described on page 8 and page 9 above. The data part of the sub-function must always be in paranthesis. Below are a few more examples...

"currentYear(1999)"... will set the year to 1999.
"setLatitude(20.0)"... will set the latitude to 20.0.
"weather_cloudCumulusAmt(0.5)"... will set the cumulus clouds to 0.5.

Alternatively you can pass a number of sub-functions in one SendMessage by separating them with commas...
SendMessage("Tenkoku_SetData","currenthour(13),currentminute(24),currentsecond(30)");

In Addition you can also use SendMessage to set the current tracking camera, by passing the name of the new camera game object in the data field. Note that this label has to match the camera object exactly.
SendMessage("Tenkoku_SetData","maincamera(MyCameraObject)");

Transitioning Time from Code

Tenkoku version 1.0.6+ includes a function that allows you to transition from one time marker to another, over the course of a set time period. This can be useful in games where you want to quickly show the passage of time, or advance to a specific time mark over a predetermined timeframe.

Tenkoku_SetTransition(startTime,targetTime,duration,direction);
or **Tenkoku_SetTransition(startTime,targetTime,duration,direction,callbackObject);**

```
float duration = 5.0f;
string startTime = “”;
string targetTime = “08:30:00”;
float direction = 1.0f;
GameObject callbackObject;

private Tenkoku.Core.TenkokuModule tenkokuModule;

void LateUpdate () {
   tenkokuModule = GameObject.Find(“Tenkoku DynamicSky”).GetComponent<Tenkoku.Core.TenkokuModule>();
   tenkokuModule. Tenkoku_SetTransition(startTime,targetTime,duration,direction ,callbackObject);
}
```

The above example will perform a 5 second transition, from the present set time to 8:30 in the morning. Once the transition is set it will call the callbackObject.

Time Format

Time is passed to the function via the specific **String** format “hh:mm:ss”, with a 2-digit hour marker, followed by a 2-digit minute marker, followed by a 2-digit second marker. These markers are set in 24-hour clock format, for example “06:45:00” will advance to 6:45 in the morning, whereas a marker such as “18:45:00” will advance to 6:45 in the evening.
If the startTime variable is set to “”, then it will assume you want to start from the current time.

Direction

Direction is a float and marked as 1.0 to signify moving forward in time, or as -1.0 to signify moving backward in time.

Callback

The callback feature is optional, and will call a specific function when the transition has completed. The callback searches for a function called **CaptureTenkokuCallback()** on the game object you specify. This can be very useful to perform an action or set a specific variable once the transition has completed.