DTT
APPS WEB GAMES

SINCE 2010

Daily Rewards

Documentation | 09-08-2022

Table of Contents

1. Get started quickly

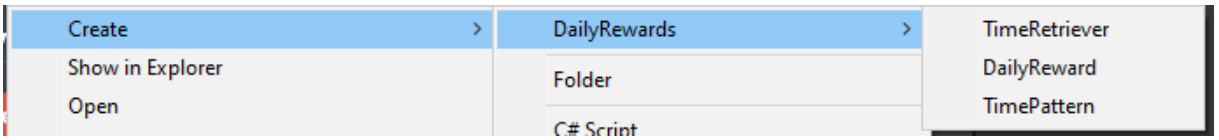
To get started, you must make three scriptable objects.

1. *A Time Pattern.*
2. *A Time Retriever.*
3. *A Daily Reward.*

Completing this section will give you a reward that can be claimed once a minute. If you wish to skip this and just use the result, see the "Every Minute" reward in the demo scene.

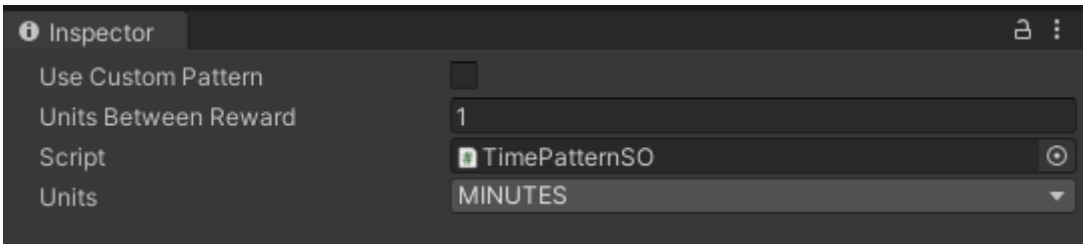
Time Pattern

1. In any folder, right-click and add the "Time Pattern" object under the "Daily Rewards" header.



The time pattern scriptable object in the create menu.

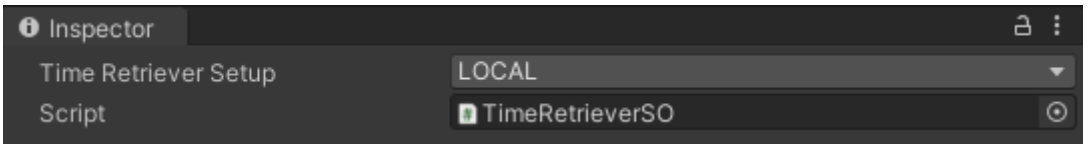
2. Open it in the inspector, and set it to the values shown below, this will create a time pattern that can be triggered every minute.



A time pattern that triggers every minute.

Time Retriever

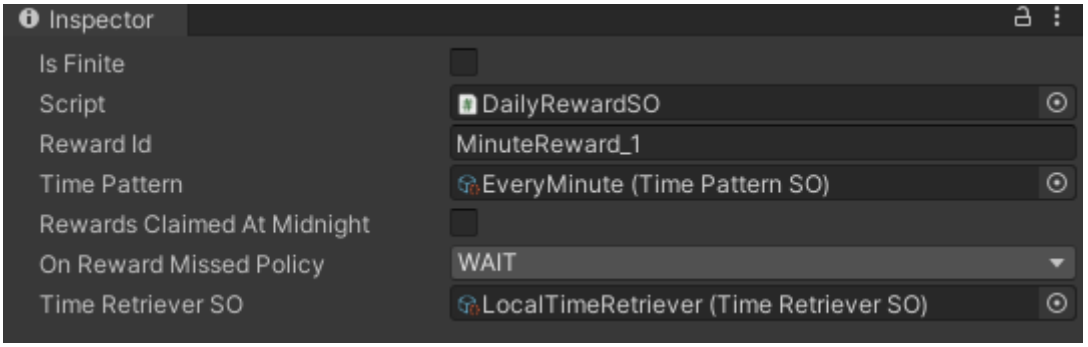
3. From the same create a menu as before, create a Time Retriever Set the time retrievers inspector values to match these shown. The time retriever will now use the system time when getting a UNIX timestamp.



A time retriever that uses the devices' local time.

Daily Reward

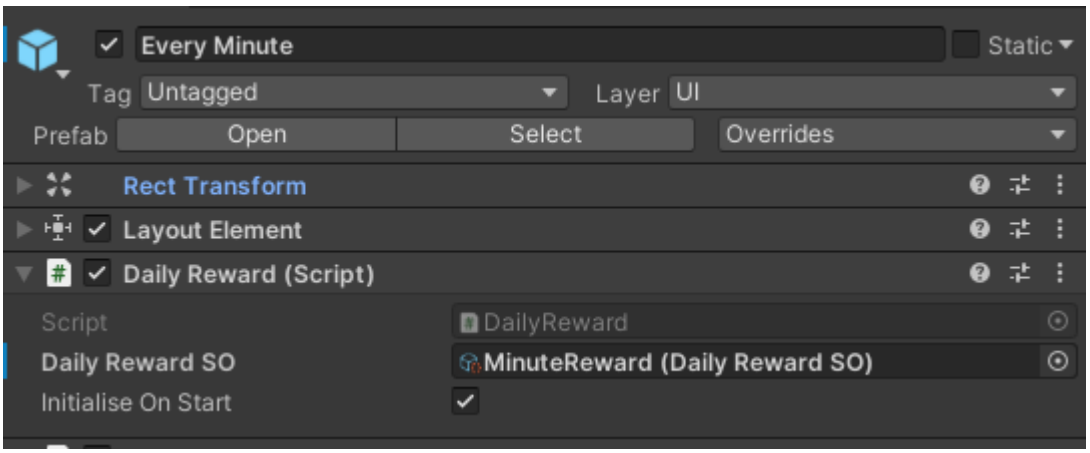
4. From the same menu, create a Daily Reward. Set the values as shown below, referencing the previously made time pattern and time retriever. You now have a reward that can be claimed once per minute.



A reward that is retrievable every minute, using local time. ("reward Id" can be changed, but a different value must be used for each reward, and the value must not be changed once in production).

Accessing In the Scene.

5. To add the scriptable object to the scene, create an empty object, add a DailyReward in the inspector, and give it a reference to the DailyRewardSO.



The daily reward is attached to the reward.

6. The daily reward scriptable object can now be accessed by referencing the daily reward mono behaviour.

Checking and Claiming.

7. Use the following methods to check and claim rewards:
    - **"CheckAvailableReward" -** Checks if a reward is available (does **NOT** claim it/affect the state of the rewards).
    - **"ClaimReward"** - Claims a reward if available.
    - **"PreviousRewards"** - Returns all previously claimed rewards.

For an example of this working and how to display the rewards to the user, check out the demo scene.

2. Introduction

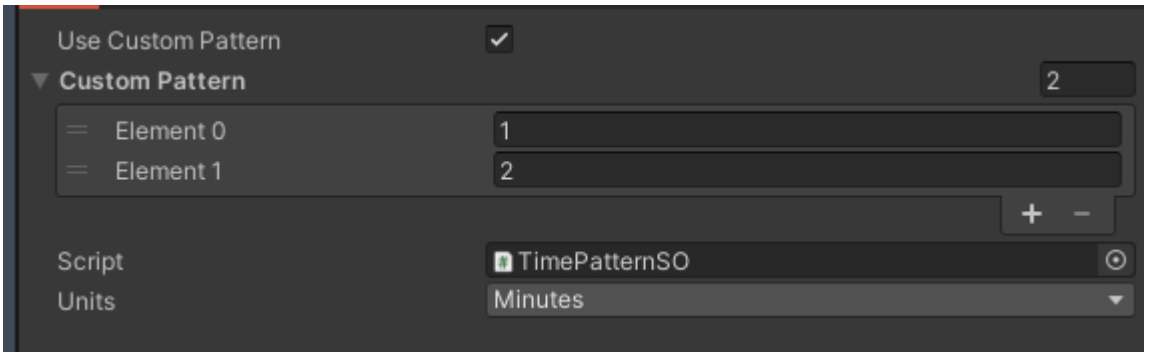
DTT Daily Rewards functions as a customizable rewards system. Allowing you to:

- Define custom time patterns to receive the reward (down to the second).
- A customizable inspector setup allows you to choose how you handle a user missing a reward.
- Use a custom or publicly available remote API for retrieving the time to make it harder for users to cheat by changing the local system time.
- Re-use time patterns across multiple rewards.
- Access the same reward in multiple areas of your application, without relying on static or singleton behaviours.

All of this can be done without writing a single line of code. speeding up your development time for any rewards system.

3. Time Patterns

A time pattern defines a repeating pattern of units of time, which a reward system can use to determine when a reward becomes available.



Time pattern which repeats a 1-2-1-2 minute pattern.

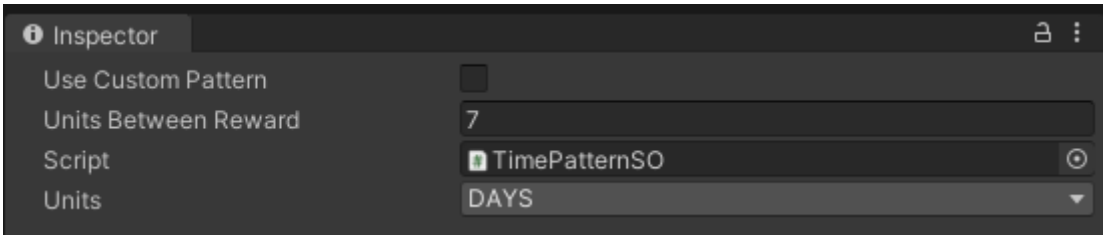
The inspector gives the following options

- **Units:** The units to measure the time pattern in (seconds, minutes, hours or days).
- **Use Custom Pattern:** Whether to use a single repeating time or a more complex pattern.
- **Custom Pattern:** If using a custom Pattern, a list that defines a recurring pattern of periods.
- **Units Between Rewards: (**If not using a custom pattern) The time period between one reward and the next becoming available.

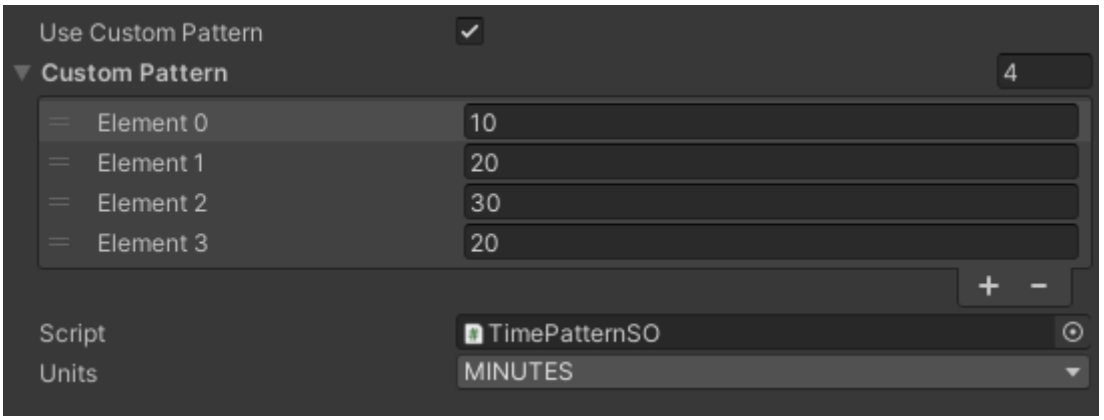
Examples

We show two examples of time patterns and how they look in the inspector.

- **Example 1** - A simple pattern that occurs once a week.
- **Example 2** - A custom pattern triggering in the sequence of 10 -> 20 -> 30 -> 20 minute intervals.
    - This will mean if the reward was claimed instantly, they would be claimed at minute 0, 10 (+ 10), 30 (+ 20), 60 (+30), 80 (+20), 90 (+10), 110 (+20), etc.



Example 1: A weekly reward

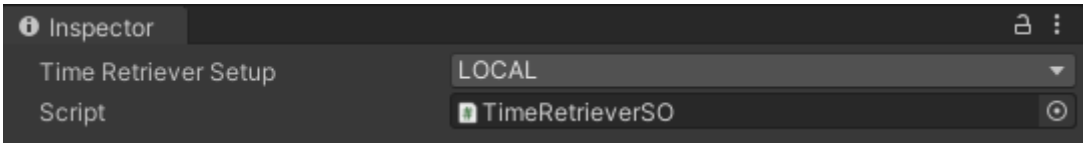


Example 2: A custom pattern

4. Time Retriever

For some setups, using the device's local time is acceptable. However, this can be easily “forged” by changing the system clock. For rewards where you have to be sure the time is accurate, we implement a time retriever, which can handle most remote APIs that give a UNIX timestamp without the need to write any code Local Time Retriever

A local time retriever should be setup up in the inspector as shown below.



This will return the system local time as a UNIX timestamp.

Remote Time Retriever

A remote time retriever has the following options:

- **Custom API URL**: The URL of the API endpoint to make an HTTP get request to.
- **Fallback Policy**: If the URL is unreachable, can we use the local time instead.
- **Is JSON**: Whether the response comes as a JSON key-value pair if not, the response is expected to be as a “raw string” that can be parsed to a UNIX time number.
- **JSON Key**: If the response is in JSON form, then which key will the UNIX timestamp be labelled with.
- **Cache Remote Call**: If we cache the response from the API. Caching means we don’t overload the server every time we need to check the time, but caching times for too long can lead to inaccurate times shown locally. A good value is a 30 seconds timeout. (cached values are incremented using local time).
- **Cache Timeout In Seconds**: How long can we use a cached time, until we must get a response from the API again.

To explain these, we work through an example of setting up a remote timestamp.

Example

We want to use the UNIX timestamp given by the URL http://worldtimeapi.org/api/timezone/Europe/Amsterdam[1]. We look at the response from the API


```
{
    "abbreviation": "CET",
    "client_ip": "95.97.240.18",
    "datetime": "2022-02-15T17:36:58.715035+01:00",
    "day_of_week": 2,
    "day_of_year": 46,
    "dst": false,
    "dst_from": null,
    "dst_offset": 0,
    "dst_until": null,
    "raw_offset": 3600,
    "timezone": "Europe/Amsterdam",
    "unixtime": 1644943018,
    "utc_datetime": "2022-02-15T16:36:58.715035+00:00",
    "utc_offset": "+01:00",
    "week_number": 7
}
```


example response received on Feb 15th 2022.

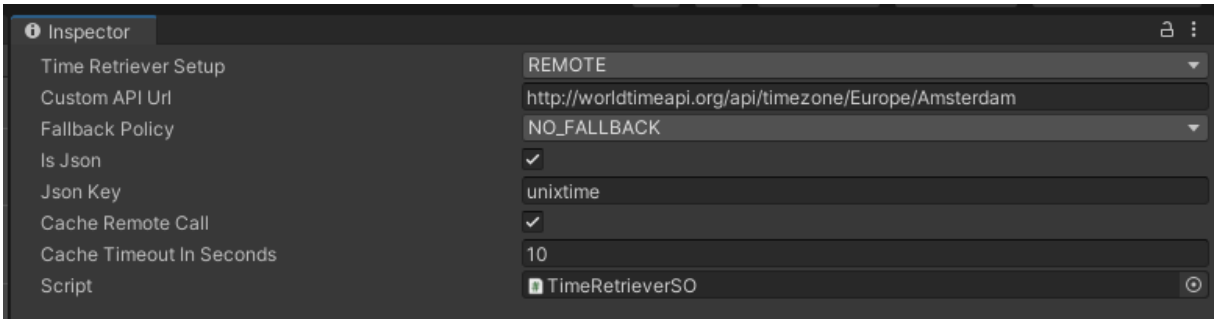
We can see that the response is in a JSON format, and the UNIX timestamp is under the key "unixtime", therefore we set *"is JSON"* = **True** and the *"JSON key"* to **"unixtime"**. We don't want to send requests too often, causing us to be rate-limited and have requests fail. There are two ways to deal with this issue:

1. Make sure we don't call *"CheckAvailableRewards"* too often/every frame.
2. **OR** enable result caching.

For this example, we decide option 2 is the best. Therefore we set "Cache Remote Call" to ***True*** and the *"Cache Timeout in seconds"* to **10**.

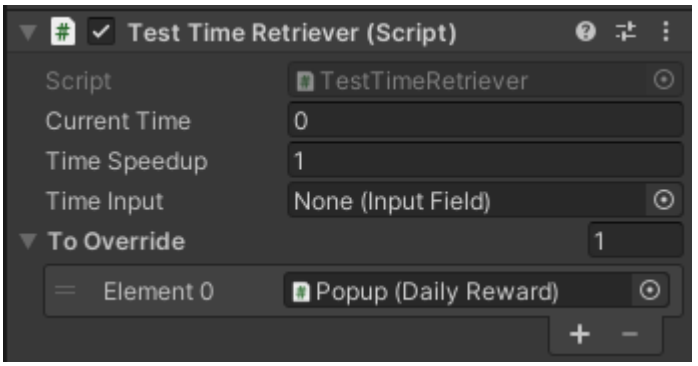
[1] This API was working on the creation of this document (Feb 15th 2022). As an external API, we have no control over this API, and cannot guarantee it being available at time of reading.

We now have a full setup for a remote API call using only the inspector values, such that the user changing their system time will not break our rewards.



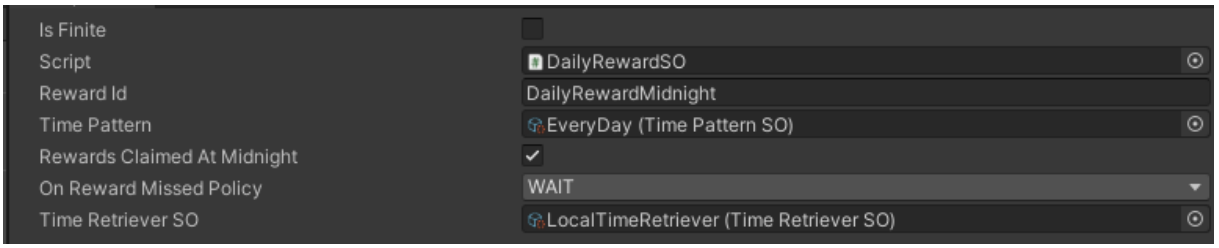
The final setup of the remote time retriever.

Debugging



Under the 'Demo/Scripts' folder, you can find a component called the **TestTimeRetriever**. This component can be used to multiply the speed of the time receivers from rewards. You simply need to put the **DailyReward** components into the override list and turn up the Time Speedup field.

5. Daily Reward SO



A reward that can be claimed once a day, any time that day.

A daily reward object brings together the time pattern and time retriever into a reward system, with customizable options to change the reward to suit your needs.

Customization Objects

The following customization options are available for daily rewards:

- **RewardID**
  - This must be a unique name, (used for storing the rewards locally).
  - This value must not be changed once the reward is in use. (or it will reset for all users).
- **Time Pattern**
  - How often the reward will become available
  - See "Time Pattern" for a more detailed explanation.
- **Rewards Claimed At midnight**
  - Makes the reward truly "daily".
  - If set, when a reward is claimed. The "claim time" is set to the beginning of the day (00:00).
  - This is used for daily rewards which are claimable after midnight, irrelevant of what time of day the last reward was claimed.
  - **DO NOT** set this to true for time patterns measured in minutes or hours, as it will break the reward (making it instantly and infinitely claimable).
- **On Reward Missed Policy**

    - This handles what to do when a reward is “missed”.
    - A reward is counted to be “missed” if it is not claimed within the time between being able to claim, and the length of the next interval.
        - For example, an hourly reward is missed if the last claimed reward was >2 hours ago.
    - The options available are:
        - **WAIT** - The next consecutive reward stays available.
            - Use when you don’t want users to miss any rewards.
        - **SKIP** - Award the reward that would be available if every previous reward had been claimed Instantly.
            - Use when you don’t want a user to fall behind, and users will miss rewards if they don’t log in frequently enough.
        - **RESTART** - The next claimed reward has its count set to 0. (users have to start again.
            - Use for systems such as streaks.
- **Time Retriever**
    - How the reward gets its value for time.
    - See “Time Retriever” for a more detailed explanation.

Using these customisations, you should be able to cover nearly all possible use cases for a daily reward.

(Advanced) Custom Value Retrievers and Reward Storers.

If you want to implement your own system of storing the rewards or to replace the time with a different value (such as score, number of invites, etc.) the reward object uses interfaces to make this possible. This allows for expanding the functionality of the rewards, some possibilities are outlined below:

1. Storing the rewards remotely, so they can be synced across devices.
2. Replacing the time retriever with a score retriever. So that rewards can be given as a user score increases.
3. Replacing the time retriever with an invite count. To reward users

6. API

To Interact with the Daily Rewards package, we recommend interacting only with the DailyReward (to access its referenced DailyRewardSO).

DailyReward is a mono behaviour wrapper that enables access to the scriptable object with the following methods and properties;

- **Reward Instance** - Returns the attached DailyRewardSO.
- **Initialise** - Completes first time setup of the daily reward object, loading them from the given reward Saver.
    - Can be used to override with custom reward savers and time retrievers for bespoke functionality.

The DailyRewardSO object has the following properties and methods, allowing you to view and interact with the reward system.

Core Functionality

The core functionality methods below are the minimum needed to use a reward system fully.

- **Initialise** - Completes first time setup of the daily reward object, loading them from the given reward Saver.
    - Can be used to override with custom reward savers and time retrievers for bespoke functionality.
- **Check Available Reward** - Checks if any reward is available. Will inform you when the next reward is available if none are available.
- **Claim Reward** - Attempts to claim any available reward.
- **Earned Rewards** - Returns the most recent (or all) rewards earned.

Helper Functionality

Helper functions give insight into the state of the reward system

- **LoadState** - Returns the current load state of the reward.

- **Loaded** - Returns a boolean value indicating whether the reward has loaded successfully or not.
    - Accessing the reward before it is loaded can lead to incorrect results being returned.
- **Override Time Retriever** - Overwrite the time retriever set in the inspector.
- **Current Streak** - Returns how many consecutive rewards the user has earned.
- **Current Reward Wait Time** - Returns the time period of the next available reward.
- **Reset Rewards** - Resets the reward, clearing all stored rewards.

6. Known Limitations

- Local time can be “forged” by changing the system time.
    - To avoid this, use a remote API with fallback not allowed.
- Player prefs are editable by the users who have a high level of technical knowledge and enough effort. allowing them to fake earned rewards.
    - To avoid this, create a custom store that holds the rewards on a remote database.

7. Support and feedback

If you have any questions regarding the use of this asset, we are happy to help you out. Always feel free to contact us at:

unity-support@d-tt.nl

(We typically respond within 1-2 business days)

We are actively developing this asset, with many future updates and extensions already planned. We are eager to include feedback from our users in future updates, be they 'quality of life' improvements, new features, bug fixes or anything else that can help you improve your experience with this asset. You can reach us at the email above.

Reviews and ratings are very much appreciated as they help us raise awareness and to improve our assets.

DTT stands for Doing Things Together

DTT is an app, web and game development agency based in the centre of Amsterdam. Established in 2010, DTT has over a decade of experience in mobile, game, and web based technology.

Our game department primarily works in Unity where we put significant emphasis on the development of internal packages, allowing us to efficiently reuse code between projects. To support the Unity community, we are publishing a selection of our internal packages on the Asset Store, including this one.

More information about DTT (including our clients, projects and vacancies) can be found here:

https://www.d-tt.nl/en/