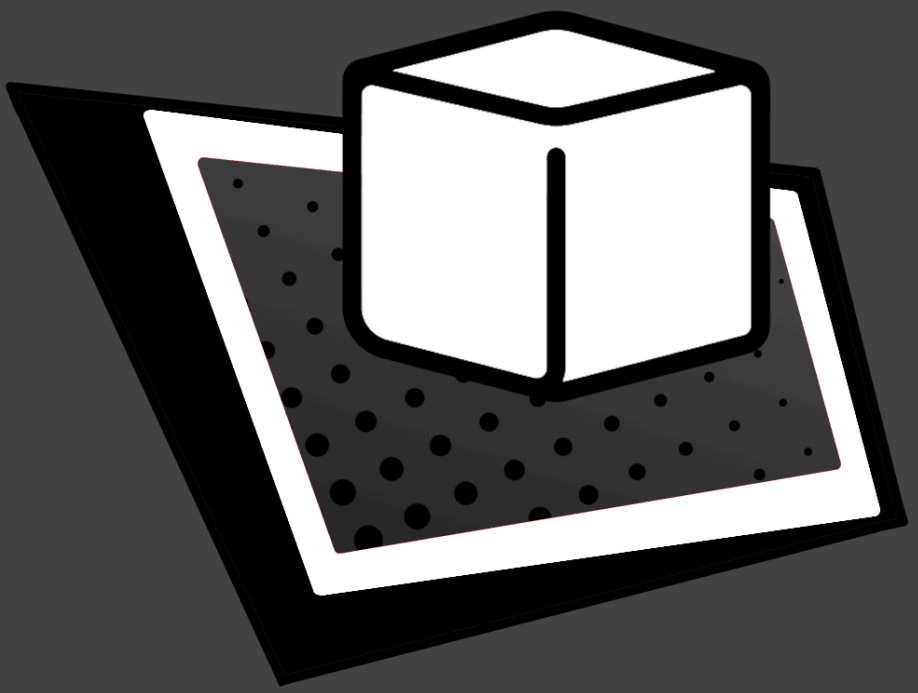


DOCUMENTATION

Summary

Asset Version	0.6.x or newer
Unity Version	2020.3.x or newer
Notion API Version	2022/06/28 (V1)
Price	FREE
Revision	1
Last Updated (Y/M/D)	2025/06/26

Notion to Unity is a tool to import Notion databases into scriptable objects so the data can be used in projects. Please note this is a tad experimental, there may be issues or edge cases that are not covered. Updates will also be slow/infrequent.

- Download databases of any size.
- Apply sorting and filters to data to order it just as it is in a Notion database view.
- Automatic parsing of data into their field types.
- Support for most useful Notion data properties.
- Automatic API key removal on build creation for security.
- System to reference assets in code without a direct inspector reference.

Contents

Supported Properties

Any string convertible type should also support JSON for custom classes, but the mileage may vary. Best to just store raw data in these assets and convert the data with an override to the PostDataDownloaded() method in the Notion Data Asset.

Note that rollups are supported only when they show a property that is otherwise supported below:

Property type (Notion)	Conversion types supported (Unity)
Title	- string
Text	- string - NotionDataWrapper(GameObject(Prefab)/Sprite/AudioClip) - List/Array of (string, int, float, double, bool)
Number	- Int - Float - Double - Long - Short - Byte - etc.
Toggle	- bool
Single-Select	- string - enum
Multi-Select	- List/Array of (string) - enum (flags)
Date	- SerializableDateTime (Wrapper class for DateTime)

Rollup	- Any supported Notion type in this table as the viewed data.

Installation & Updating

GitHub (Recommended)

For GitHub you'll need to navigate to the latest release and download the package provided in the assets for the release.

It is recommended to use the repo URL in the package manager in Unity as it keeps the assets' files in a safe space and allows for easy updating. The URL to add it:

https://github.com/CarterGames/NotionToUnity.git

If you want to use a standard unity package instead, there will always be a .unitypackage file provided in each release. Then you just import that package into your project. You can also just download the source and copy/paste it into your project for a similar result.

You can also use the pre-release branch to get the latest updates before they are officially released. Use this at your own risk.

https://github.com/CarterGames/NotionToUnity.git#pre-release

Itch.io

For itchio, just download the asset .unitypackage and import it into your project.

Updating The Asset

Most updates will be fine to import over an existing install so long as the version you are using is in the same major version. A major version is the first number in the version number. So the current one is **0**.x.x. If that changes, it is recommended to perform a clean install instead. Some updates will require a clean install to function properly. See the release notes for the new version to see if this is necessary.

Clean Install

A clean install is where you remove all files from the asset and then import the new files into the project. It’s recommended when elements break due to updating the asset.

Setup Guide

Notion Setup

You need to make an integration in order for the downloading to work. You can make an integration here. The steps to follow are:

- Make a new integration with the **New integration** button.
- Select the workspace the integrations can access. This should be the workspace the database(s) you want to download are in.
- Give the integration a name & continue to the next page.
- Navigate to the **Capabilities** tab from the sidebar and ensure the integration has read access. You can disable the rest as you only need the readability.
- Navigate to the **Secrets** tab and copy the key for use in Unity.

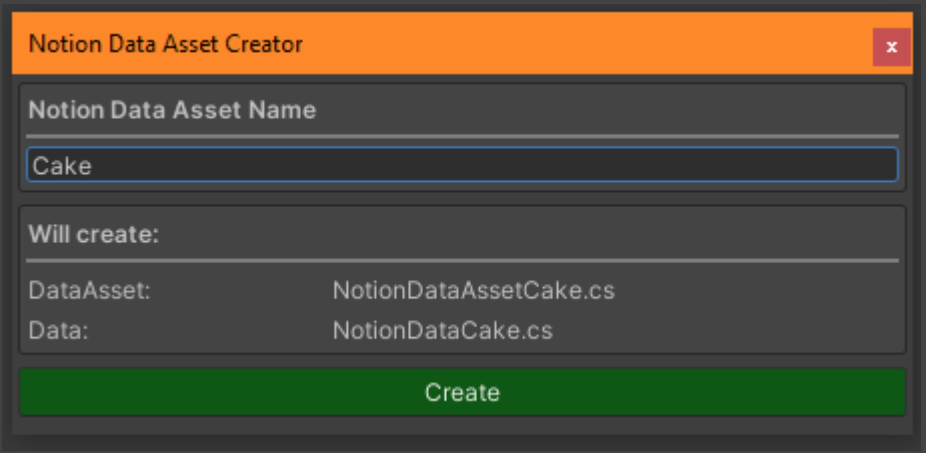
Once done you can then enter Notion and add the integration to one or multiple pages to allow the Notion API to access the data. This is done from: **... > Manage Connections >** . Find and add your integration from the options listed. If you don't see the integration you just made listed, close & open Notion and try again.

Unity Setup

In Unity you use a **Notion Data Asset** to store the data. This is just a scriptable object which has a custom inspector to aid with the data download. Each instance you make will consist of the data asset, a scriptable object class and the data class which holds the data structure for the data asset to store. There is a tool to make these for you which can be found under **Tools > Standalone > Notion Data > Asset Creator**

Asset Creator

The asset creator is an editor window that handles creating the classes for a **Notion Data Asset**. You just enter a name for the class you want to make and press **Create** when ready. You'll be able to see a preview of what the classes will be called before your press the **Create** button. Once pressed you'll be able to select where you want to save each newly generated class. It's best to keep them together in the same directory for ease of use.

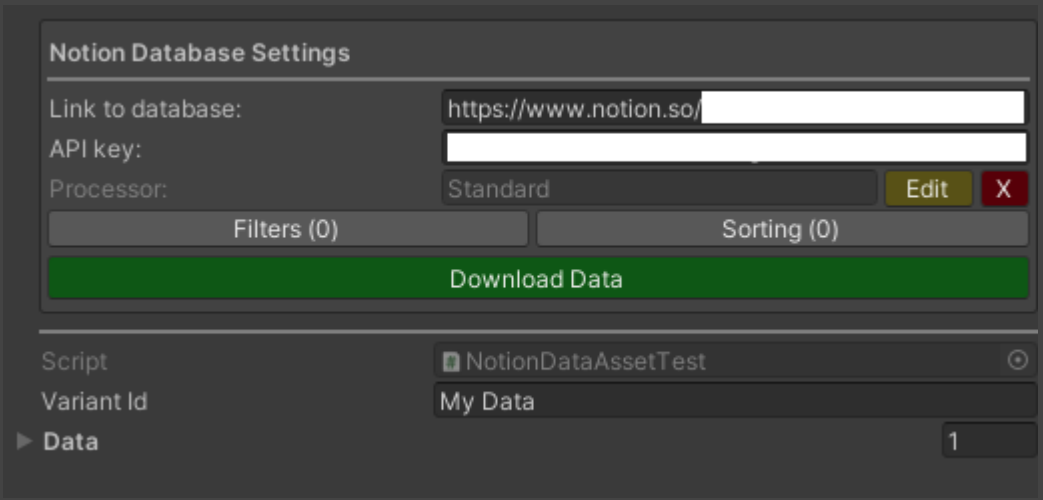


Once set up you'll just need to write your data class to have the fields you want. An example of a Persona healing skill data class:

```
[Serializable]
public class DataHealingSkills
{
    [SerializeField] protected string skillName;
    [SerializeField, TextArea] protected string desc;
    [SerializeField] protected NotionDataWrapperSprite icon;
    [SerializeField] protected SkillType type;
    [SerializeField] protected ActionTarget target;
    [SerializeField] protected SkillCost cost;
    [SerializeField] protected NotionDataWrapperPrefab effect;
    [SerializeField] private float power;
    [SerializeField] private StatusAilment cureAilments;
    [SerializeField] private bool canRevive;
}
```

Then just fill the fields on the data asset (make one from the **CreateAssetMenu** if you haven't already):

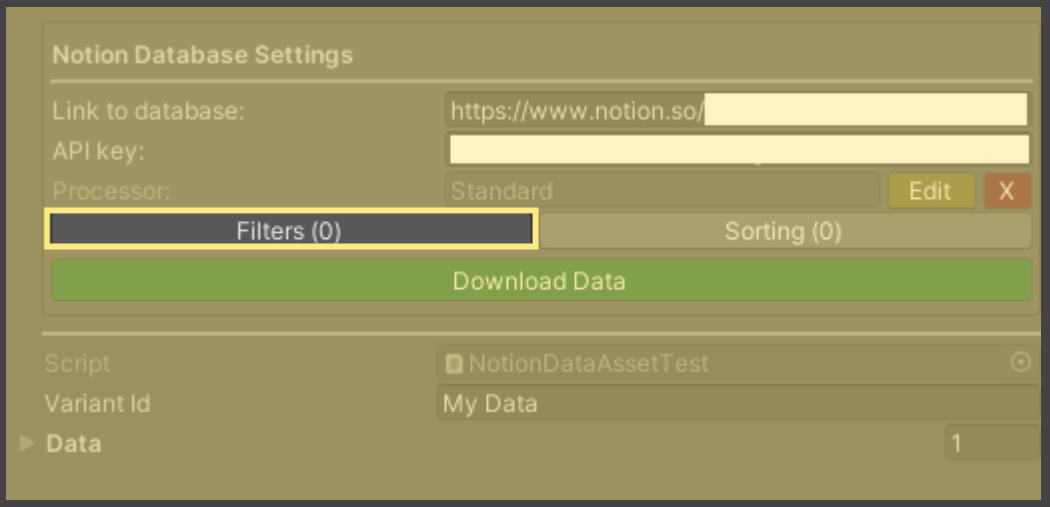
- **Link to database:** Copy and paste the link to the database in Notion.
- **API key:** The secret key of your Notion integration to use to download data with, as setup earlier.
- **Processor:** Defines how the data is handled in Unity once downloaded. You can write custom ones to fit your use-cases or use the standard one which will work for basic 1-2-1 data where there is no conversion or class setup needed. Read more on custom processors [**here**]



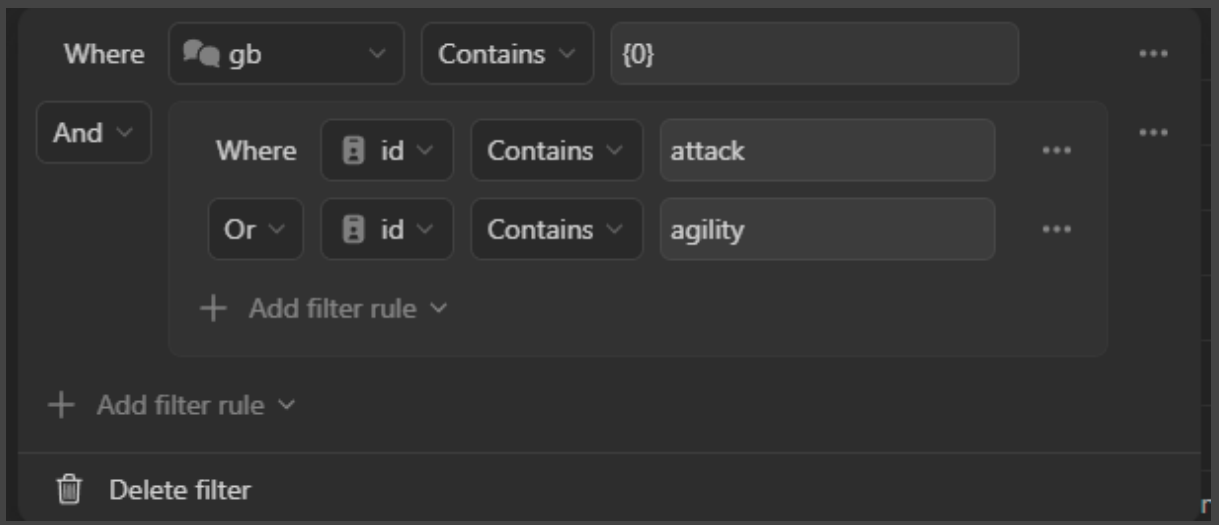
Then press the download button. If all goes well you'll see a dialogue stating so. If it fails you should see the error in the console.

Filters

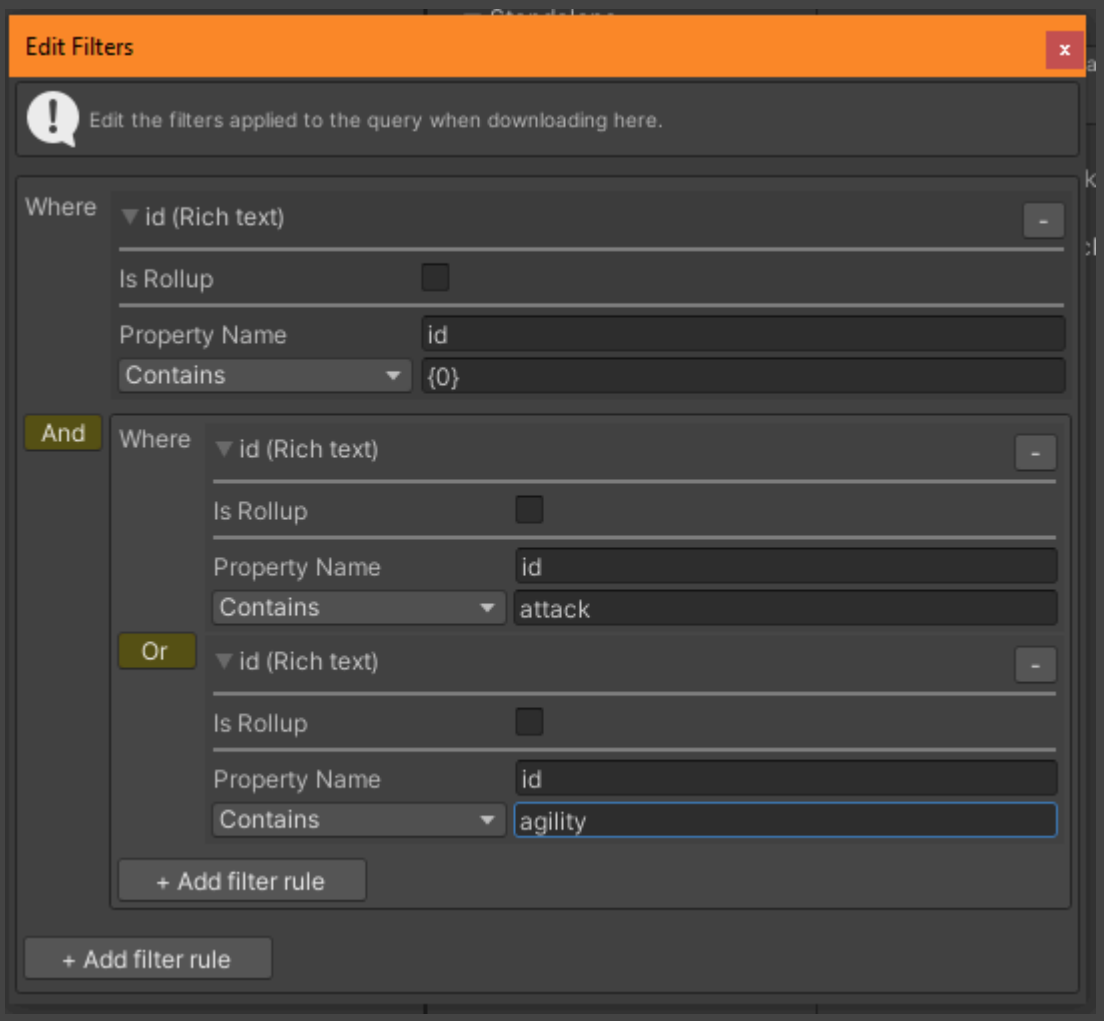
You can apply filters to the download requests by using the filters window. This window more or less mimic's Notion's filters GUI. This setup supports the property types the system currently supports reading. The only notable difference is with rollup support. It is supported, but you'll have to use the type the rollup is displaying and then define it as a rollup of that type for it to work.



Example (Notion)

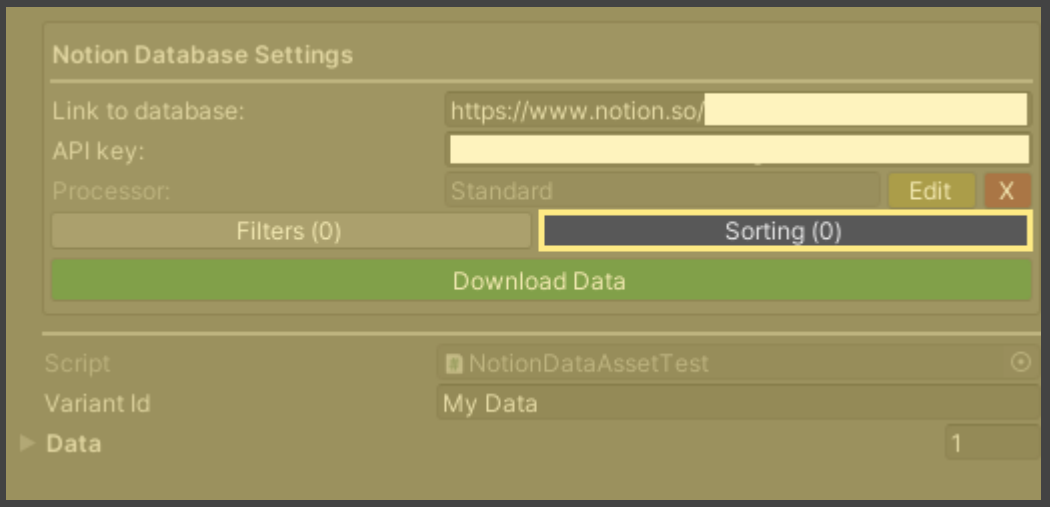


Example (Unity)

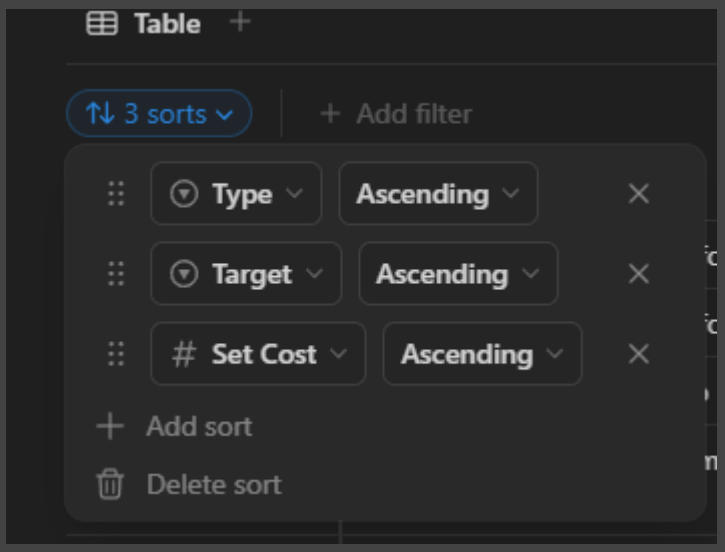


Sorting Properties

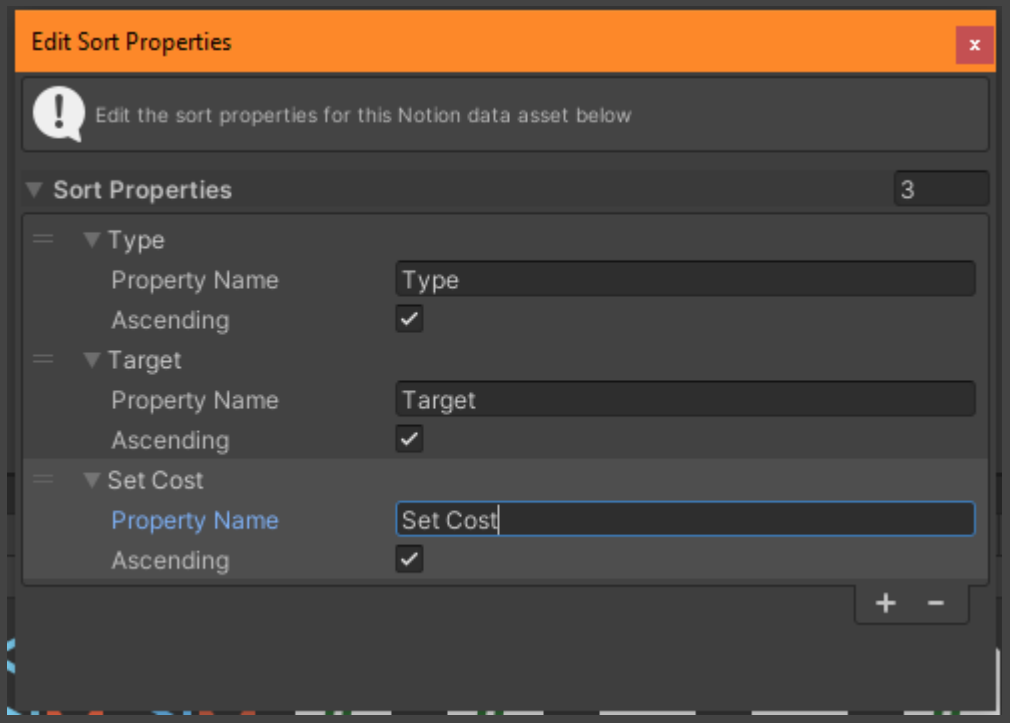
You can apply sorting properties to your download requests by adding them to the sort properties list in the inspector. The text for each entry is the Notion property name you want to sort by, with the checkbox set to if you want to sort ascending for that property. The order of the sort properties in the list defines the order they are used, just like in Notion.



Example (Notion)



Example (Unity)



Wrapper Classes

Some data needs a wrapper class to assign references. This is provided for GameObject prefabs, Sprites & AudioClips should you need it. They are assigned by the name of the asset when downloading the data. To add your own just make a new class that inherits from **NotionDataWrapper** and override the **Assign()** method to parse a string value in Notion into the desired type in Unity. If the logic to assign the values uses editor logic make sure you add the **#if UNITY_EDITOR** scripting define to all editor logic so the project will compile for builds.

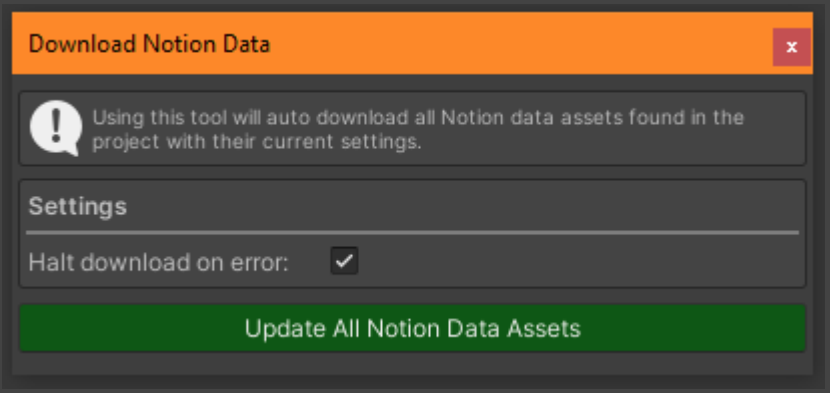
Post Download Logic

You can also manipulate the data you download after receiving it by writing an override to the method called **PostDataDownloaded()** on the **NotionDataAsset**. Note if you need to run editor logic, make sure it is in a **#ifdef**. An example below:

```
#if UNITY_EDITOR
    protected override void PostDataDownloaded()
    {
        skillLookup = new SerializableDictionary<string, DataHealingSkills>();

        foreach (var data in Data)
        {
            // Runs a method called PostDownloadLogic() on the data class instance.
            data.GetType().GetMethod("PostDownloadLogic", BindingFlags.NonPublic | BindingFlags.Instance)
                    ?.Invoke(data, null);

            // Adds the data class to a lookup for easier use at runtime.
            skillLookup.Add(data.Name, data);
        }

        // Saves the changes to the scriptable object.
        UnityEditor.EditorUtility.SetDirty(this);
        UnityEditor.AssetDatabase.SaveAssets();
    }
#endif
```

Update all assets

You can download all data assets in one process through an additional editor window. The window can be found under **Tools > Carter Games > Standalone > Notion Data > Update Data**. The window has the option to halt the downloading of assets if an error occurs, by default this true. To download all assets in the project, just press the download button and wait for the process to complete.



Scripting API Info

Assemblies

If you are using assemblies for your code base, you'll need to reference the audio manager assemblies to access the API of the asset.

Editor > CarterGames.NotionData.Editor
Runtime > CarterGames.NotionData.Runtime

The asset also has some shared libraries between assets. If you need to access these, you can do so from these assemblies:

Shared Editor > CarterGames.Shared.NotionData.Editor
Shared Runtime > CarterGames.Shared.NotionData

Namespace

The main namespace for the asset is CarterGames.NotionData

Custom Notion Database Processors

Processors are what handle converting the data received from Notion into something we can use in Unity. The standard processor is always available for use and will attempt to process each column of the database that was downloaded into fields that match those column names 1-2-1. For simple use-cases, this will be enough. If you need extra functionality or the ability to merge properties into collections etc you'll need to make your own processor.

API explanation

```
// Contains the result of the download from Notion
NotionDatabaseQueryResult result

// Is a collection of all the rows on the database that was downloaded.
result.Rows

// In each row there is a data lookup which can be used to search for specific
// columns by their string name as shown in Notion.
result.Rows[0].DataLookup


// From here you'll have a NotionProperty type to use.
// This can be converted to a specific type like Date, Number, Select etc.
// Which formats the json value into a c# equivalent.
// You will need to do extra work to parse them fully though.
// For enums you'll need to parse them into their specific types to read them as an example.

// Converts the property to a rich text type property and surfaces the value as a string for
use.
result.Rows[0].DataLookup["myproperty"].RichText().Value


// Other types as an example.
result.Rows[0].DataLookup["myproperty"].Date().Value
result.Rows[0].DataLookup["myproperty"].Number().Value


// Attempts to convert to the entered type (used in the standard processor, can be used in
any custom ones as well).
result.Rows[0].DataLookup["myproperty"].TryConvertValueToType<T>(out var result)
```

An example of a custom processor:

```
// Converts data into a list of localization entries
// which are a key string value string class.
[Serializable]
public sealed class NotionDatabaseProcessorLocalization : NotionDatabaseProcessor
{
        /// <summary>
        /// Parses the data when called.
        /// </summary>
        /// <param name="result">The result of the download to use.</param>
        /// <returns>The parsed data to set on the asset.</returns>
        public override List<object> Process<T>(NotionDatabaseQueryResult result)
        {
            var list = new List<object>();

            foreach (var row in result.Rows)
            {
                var entries = new List<LocalizationEntry<string>>();

                foreach (var k in row.DataLookup.Keys.Where(t => t != "id"))
                {
                    var valueData = row.DataLookup[k];
                    entries.Add(new
LocalizationEntry<string>(row.DataLookup[k].PropertyName, valueData.RichText().Value));
                }

                list.Add(new LocalizationData<string>(row.DataLookup["id"].RichText().Value,
entries));
            }

            return list;
        }
}
```

The database it reads from:

Support Skill Messages

Table +

+ Add filter

id	gb
support_buff_attack_increase	{0} Attack up!
support_buff_attack_decrease	{0} Attack down!
support_buff_attack_reset	Attack reverted!
support_buff_defense_increase	{0} Defence up!
support_buff_defense_decrease	{0} Defence down!
support_buff_defense_reset	Defence reverted!

Accessing Notion Data Assets

You can reference the assets as you would a normal scriptable object in the inspector. Or you can use the **NotionDataAccessor** class in the project to get them via code. Each **NotionDataAsset** has a variant id in the inspector. By default the variant id is a new GUID on creation. You can change this to help you identify a single instance of assets of the same type as another. Some example usage below:

```
private void OnEnable()
{
    // Gets the first asset of the type found.
    var asset = NotionDataAccessor.GetAsset<NotionDataAssetLevels>();

    // Gets the asset of the matching variant id.
    asset = NotionDataAccessor.GetAsset<NotionDataAssetLevels>("MyAssetVariantId");

    // Gets all of the assets of the type found.
    var assets = NotionDataAccessor.GetAssets<NotionDataAssetLevels>();
}
```

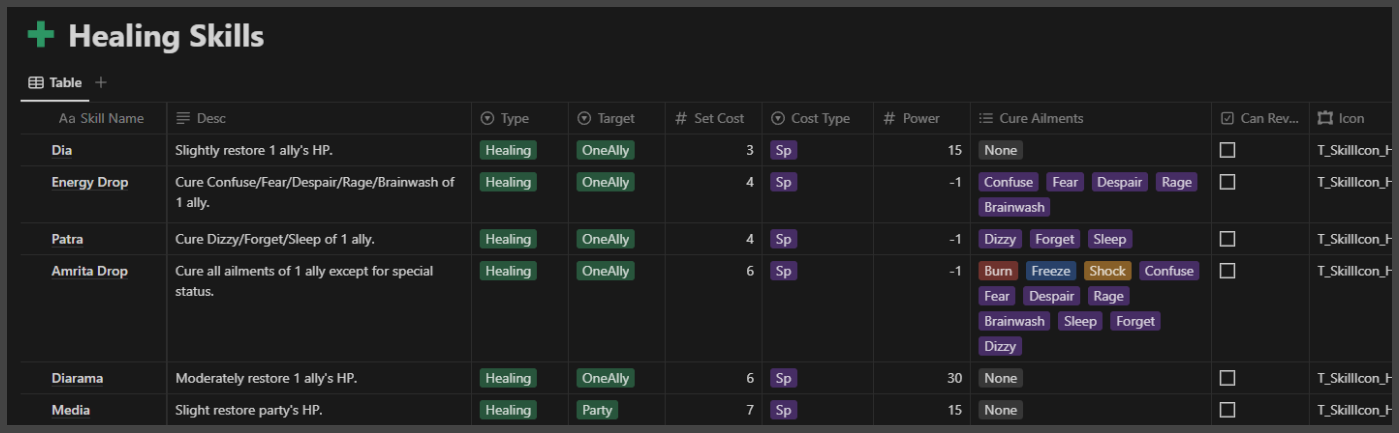
Security of secret keys & URLs

In builds you will not have an issue with these. All urls & keys are removed from the **NotionDataAsset's** before a build is made and returned when the build is completed. They will remain in the project if you are using source control which will be the only case where they are saved otherwise. Keeping the Notion integration to read-only will make it fairly safe as is.

Example

Below is an example using the system to store data for Persona 5 healing skills for persona's

Example (Notion):
A database of all the skills for healing:

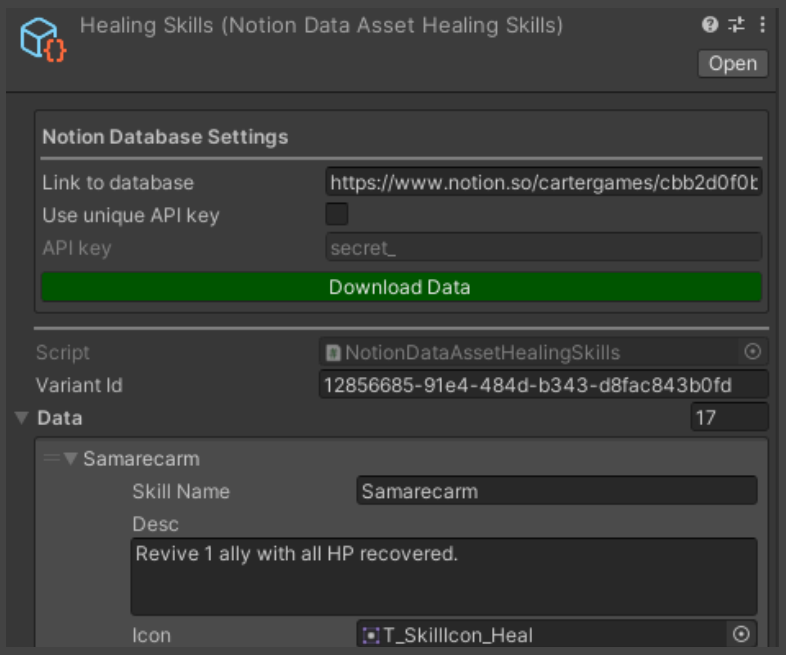


Healing Skills

Table

Aa Skill Name	Desc	Type	Target	# Set Cost	Cost Type	# Power	Cure Ailments	Can Rev...	Icon
Dia	Slightly restore 1 ally's HP.	Healing	OneAlly	3	Sp	15	None	☐	T_SkillIcon_H
Energy Drop	Cure Confuse/Fear/Despair/Rage/Brainwash of 1 ally.	Healing	OneAlly	4	Sp	-1	Confuse Fear Despair Rage Brainwash	☐	T_SkillIcon_H
Patra	Cure Dizzy/Forget/Sleep of 1 ally.	Healing	OneAlly	4	Sp	-1	Dizzy Forget Sleep	☐	T_SkillIcon_H
Amrita Drop	Cure all ailments of 1 ally except for special status.	Healing	OneAlly	6	Sp	-1	Burn Freeze Shock Confuse Fear Despair Rage Brainwash Sleep Forget Dizzy	☐	T_SkillIcon_H
Diarama	Moderately restore 1 ally's HP.	Healing	OneAlly	6	Sp	30	None	☐	T_SkillIcon_H
Media	Slight restore party's HP.	Healing	Party	7	Sp	15	None	☐	T_SkillIcon_H

Example (Unity):
The downloaded data in Unity:



Support

Need extra help?

If you need additional help with the asset, you can contact me through email. Either directly or via the contact form on the Carter Games website.

Contact Form: https://carter.games/contact/
Email: hello@carter.games

Found a bug?

Please report any issues you find to me either via the bug report form on the Carter Games website or through an issue on GitHub. I'll try to fix these as soon as I can once reported. If I don't acknowledge your bug report, feel free to give me a nudge via email just in-case I haven't received the notification.

Bug Report Form: https://carter.games/report/
Github Issues: https://github.com/CarterGames/NotionToUnity
Email: hello@carter.games