

What is Sheet Codes?	2
Getting started	3
Add Sheet	4
Sheet Codes main window	5
Add Row	5
Add Column	6
Delete Sheet	8
Show Enum Value	8
Save	8
“Sheet Name” dropdown	8
Array Edit Panel	8
Editing a Row	9
Editing a Column	9
Accessing your data	10
Editing your data at runtime	10
Generated Scriptable Objects	11
Loading sheets manually	11
Updating from version 1.0, 1.1 to version 1.2	11
Updating to version 1.3	12
Updating Sheet Codes or Upgrading from Sheet Codes Free	12
Tool access prevention	13
Compiler errors outside Sheet Codes	13
Compiler errors in Sheet Codes	13
Extra features	14
Contact	14

What is Sheet Codes?

Sheet Codes is a data management package for Unity. By using a fully custom-built editor window, developers can create and manage data in the style of Excel or Google Sheets, right here in Unity.

The Sheet Codes tool supports almost any data type with no added work. This includes data types you create or data present in libraries within your Unity project.

Supported types include:

- Long
- Int
- Short
- Byte
- Float
- Double
- String
- Boolean
- Color
- Any Enum in your project.
- Any Type derived by `UnityEngine.Object` in your project.
- Array of any of the above types

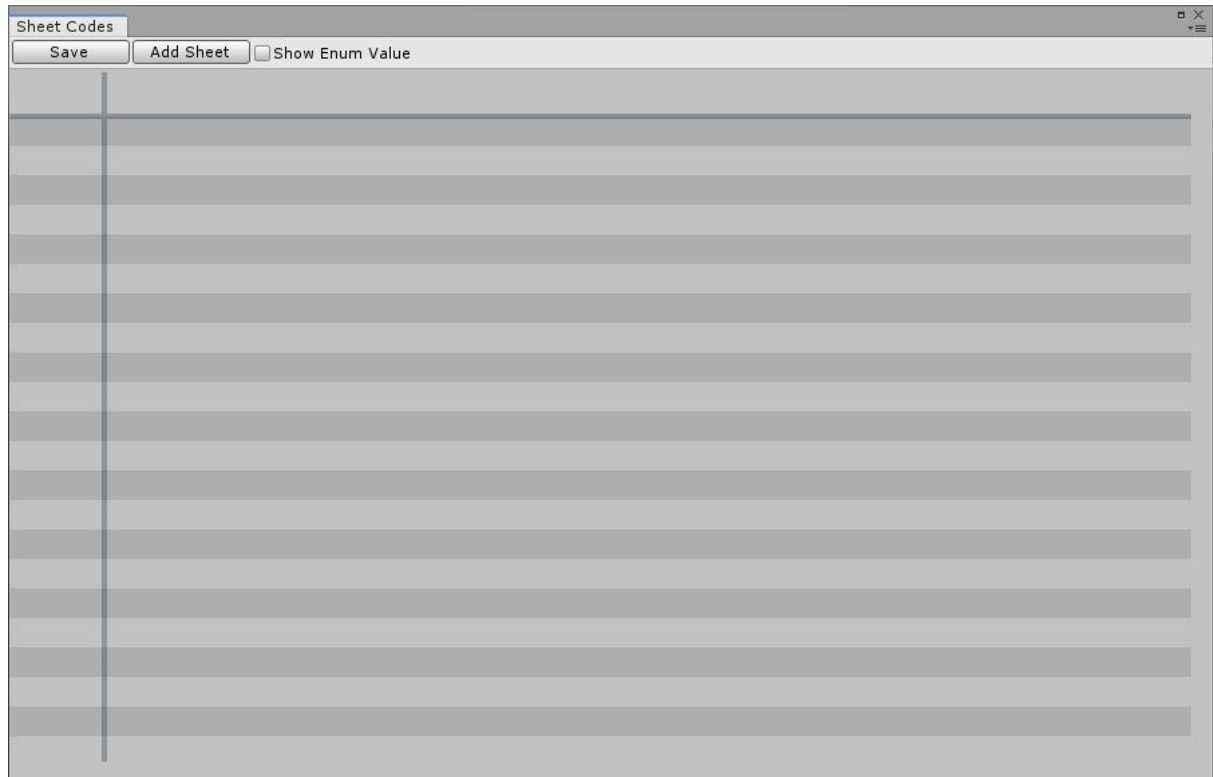
`UnityEngine.Object` is the Base class for all objects Unity can refer to, meaning this tool supports anything, such as Sprites, Animations, GameObjects, Transforms, MeshRenderers, Scenes, and much more.

The tool is supported by a full-automatically generated codebase, optimized to work with IntelliSense. Finding your data couldn't be simpler. On top of that, all data are stored as the type specified in the editor. Programmers will no longer have to cast their data!

Getting started

Open the tool by clicking on Tools → Sheet Codes. You will be greeted by a window with 2 buttons in the top left corner of your screen:

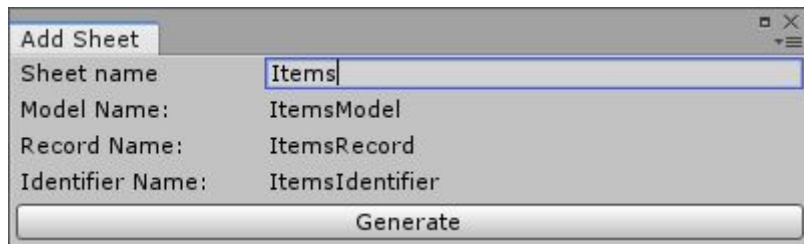
1. Save
2. Add Sheet



Add Sheet

Press “Add sheet” to open a new window. This window enables you to specify the sheet name and the generated script names. The tool generates 3 scripts for each sheet:

- 1: *SheetName*Model
- 2: *SheetName*Record
- 3: *SheetName*Identifier



The generated scripts:

- *SheetName*Model is used to access your data in code.
- *SheetName*Record is all information on a single row in your sheet.
- *SheetName*Identifier is an Enum that contains a value for each row and is used to specify which Record you wish to retrieve from the Model.

If you happen to already have a class of the same name: do not worry. The scripts are encapsulated in a namespace preventing any compiler errors.

All 3 scripts will be saved in a folder named:
SheetCodes/Scripts/GeneratedCode/*SheetName*.

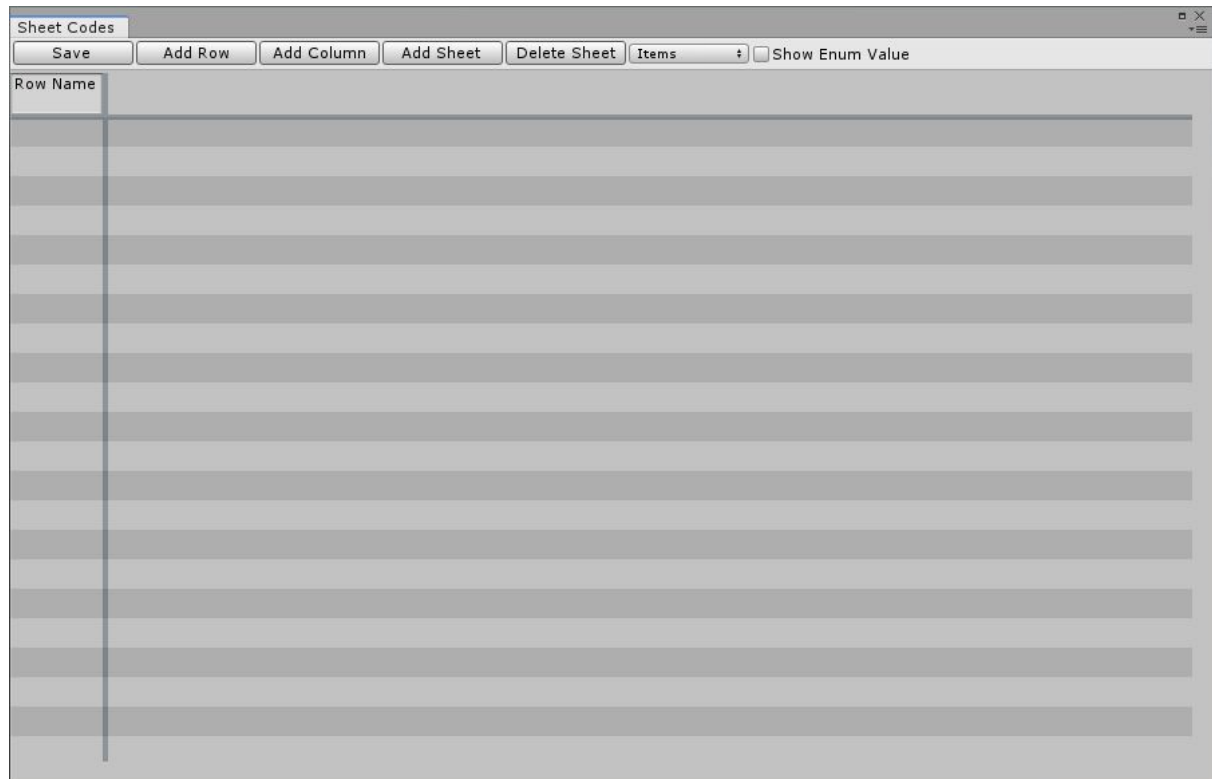
In case one of the following occurs, an error message will be displayed, and the sheet cannot be created:

- the name of the sheet already exists.

After pressing the button “Generate”, the window “Create Sheet” will close, and the main window will have new buttons.

Sheet Codes main window

If there is at least one sheet, the main window looks as follows:



From here you can start filling in your sheet by adding rows and columns. Each row represents a record and each column represents a Property in that record.

Add Row

By pressing “Add Row”, a new window will appear. This window shows 3 fields: Row Name, Enum Value, and Index.



The fields are:

- Row Name is the name you give to your row. “Row Name” cannot contain the characters “ and \.
- Enum Value is the name of the generated Enum for this row. It is automatically filled in using CamelCase, which is based on your “Row Name”. However, you can edit the value to be whatever you want it to be. “Enum Value” is limited to only contain numbers, letters, and underscores. In addition, “Enum Value” cannot begin with numbers.

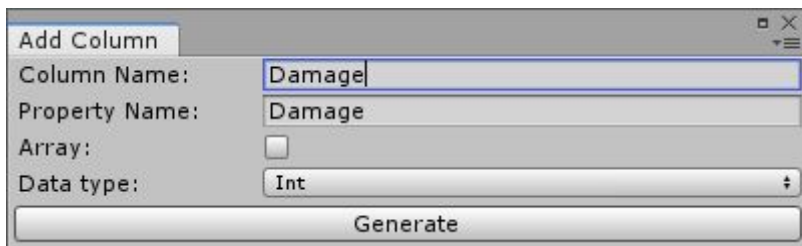
- Index is the value given to this Enum. This field is automatically filled in with 1 plus the current highest Index in the sheet. However, you can edit the Index to be whatever you want it to be. “Index” must be higher than 0 and cannot match “Index” of another row in this sheet.

If any of the following prerequisites are met, an error message will be displayed, and you cannot create a row:

- “Row Name” is empty.
- “Row Name” matches “Row Name” of another row in this sheet.
- “Enum Value” is empty.
- “Enum Value” matches “Enum Value” of another row in this sheet.
- “Index” is not higher than 0.
- “Index” matches “Index” of another row in this sheet.

Add Column

By pressing “Add Column” a new window will appear, containing multiple fields which are based on the selected Data Type.



The screenshot shows a dialog box titled "Add Column". It has four input fields: "Column Name" (containing "Damage"), "Property Name" (containing "Damage"), "Array" (with an unchecked checkbox), and "Data type" (a dropdown menu showing "Int"). At the bottom is a "Generate" button.

The fields are:

- Column Name is the name you give your column and should be one that describes the contents of your column. Column Name cannot contain characters “ and \.
- Property Name is the name of the property that will be generated in the *SheetName*Record script. It is automatically filled in using CamelCase based on your “Column Name”. However, you can edit it to be whatever you want it to be. “Column Name” is limited to only contain numbers, letters, and underscores. In addition, “Column Name” cannot begin with numbers.
- Array defines whether this column should be an array of the specified Data Type. Please refer to “Array Edit Panel” for more information.
- Data Type specifies what type of data this column contains.
- In the case a Data Type is selected, Enum, Component, or Reference require the completion of additional fields.
 - Data Type Enum requires you to select an Enum Type from a dropdown list. By default, all Enums found in your non-editor assembly are located in the dropdown list. The field “Filter text” can be used to narrow down the entries in the dropdown list to help find the desired Enum Type.

```

1  public enum EquipmentSlot
2  {
3      Arms,
4      Hands,
5      Legs,
6      Feet,
7      Neck,
8      Finger,
9      Head,
10     UpperBody,
11     LowerBody
12 }

```

Add Column
 Column Name: Equipment Location
 Property Name: EquipmentLocation
 Array: ☐
 Data type: Enum
 Filter:
 Enum Type: EquipmentSlot
 Generate

- Data Type Component requires you to select a Component Type from a dropdown list. The dropdown “Component Type Selection” specifies the types in the list; the field “Filter text” can be used to narrow down the entries to help find the desired Component Type.

```

1  using UnityEngine;
2
3  public class Equipment : MonoBehaviour
4  {
5  }

```

Add Column
 Column Name: Prefab
 Property Name: Prefab
 Array: ☐
 Data type: Component
 Component List: Custom
 Filter:
 Component Type: Equipment
 Generate

- Data Type Reference requires you to select a Sheet from a dropdown list. After creating the column, the records from the selected sheet can now be selected in the column’s cells.

Delete Sheet

By pressing “Delete Sheet” the currently selected sheet will be deleted. Any columns in other sheets with Data Type Reference, and which have the deleted sheet as the referenced sheet, will also be removed.

Show Enum Value

The checkbox “Show Enum Value” changes what is shown in the first column. In case the checkbox is not toggled, only the “Row Name” of the rows is shown. When the checkbox is toggled, it also shows the “Enum Value” and the “Index”. Leaving this checkbox on / off has no impact on the generated code.

Save

Because the tool generates code, the tool applies all changes after saving. When closing the main window by pressing X in the top right, a prompt will ask you whether you want to save any unsaved changes (if there are any changes).

After saving, the tool automatically closes and starts recompiling twice: the first is to create, update, and remove scripts. The second is to create the scriptable objects and fill them in. Please wait for these recompilations to finish before doing anything else.

“Sheet Name” dropdown

The current selected sheet is displayed by a dropdown next to the “Delete Sheet” button. Changing to a different sheet will update the layout. You can edit multiple sheets before saving.

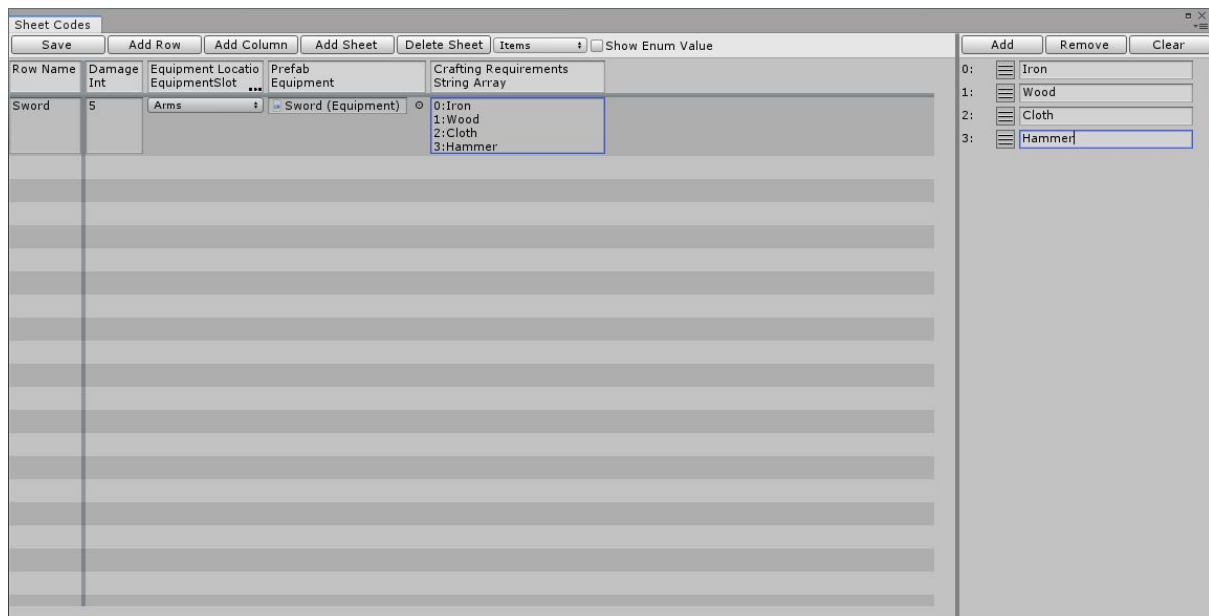
Array Edit Panel

When a column is an Array of any type, a small side panel opens when selecting any cell of that particular column. In the bottom of this side panel, 3 buttons are shown: Add, Remove, and Clear:

1. By pressing ‘Add’, you can add a single element at the bottom of the Array.
2. By pressing ‘Remove’, you can remove a single element from the bottom of the Array.
3. By pressing ‘Clear’ you can remove all elements from the Array.

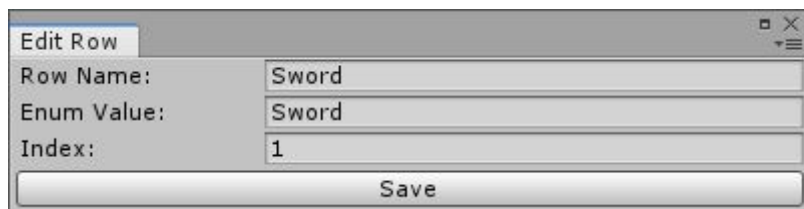
Each element in the Array is displayed with the index in front of it. This makes it easier to keep track of the index in case it has many elements.

Right next to the index is an image which can be used to click and drag elements in the Array. By right-clicking on the image, some additional options are shown to insert or remove that element.



Editing a Row

To edit any row right-click on the cell containing the row's name and press "Edit Row". A window will show similar to the one of "Add Row" with the fields already filled in. Here you can edit any of the row's properties.



Editing a Column

To Edit any column right-click on the cell containing the column's name and press "Edit Column". A window will show similar to the one of "Add Column". From here you can change anything from the generated "Property Name" to the "Data Type". When changing the Data Type it will try to change the data from cells to the new Data Type.

Examples

- A Data Type is changed from float to int. Cell values would change from 5.44 -> 5.
- A Data Type is changed from int to string. Cell values would change from 5 -> "5".
- A Data Type is changed from string to short. Cell values would change from "5" -> 5.

When working with a column of Data Type: Component it will try to automatically fill in the new cell values based on the current cell values.

Examples:

- A Data Type is changed from GameObject to Transform. Each cell will try to retrieve the Transform from the GameObject to become the new cell value.
- A Data Type is changed from Texture to Texture2D. If the cell value is also of type Texture2D the cell value will remain the same.
- A Data Type is changed from Transform to MeshRenderer. Each cell will try to retrieve MeshRenderer from their GameObject to become the new cell value.

When changing the column to be from Array to not-Array the first element of each Array is used as the new cell value.

When changing the column to be from not-Array to Array the first element of the Array will be that of the old cell value.

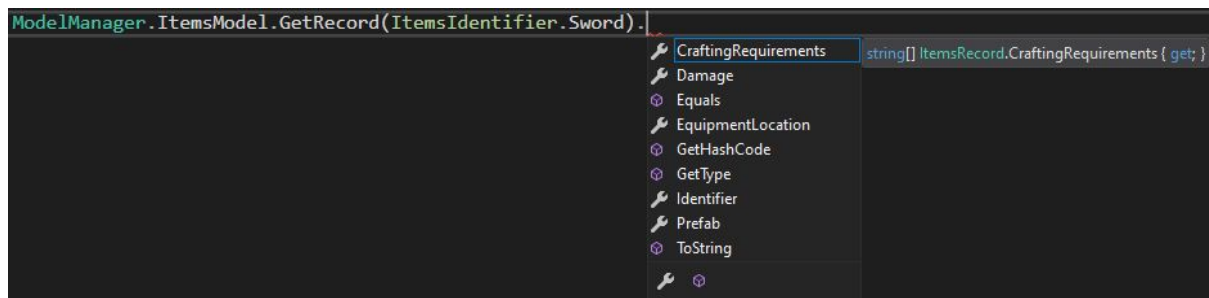
Changing the Array status works while in combination with editing the Data Type.

Accessing your data

Retrieving your data can be done in 2 ways. Each can be done at any point during runtime.

- `ModelManager.*SheetName*Model.GetRecord(*SheetName*Identifier.YourRow)`
- `*SheetName*Identifier.YourRow.GetRecord()`

Each of these methods will return a record. From there you can use IntelliSense to easily find the property you need. Properties are named based on the Property Name field when adding a column.



Editing your data at runtime

All records can be edited during runtime as long as you're still in the Unity Editor.

ScriptableObjects cannot in any way be edited outside the Unity Editor. This is a unity feature and there is no way around this.

To make changes to a scriptableobject request an editable copy of your record. You can do this in 2 ways.

- `ModelManager.*SheetName*Model.GetRecord(*SheetName*Identifier.YourRow, true)`
- `*SheetName*Identifier.YourRow.GetRecord(true)`

All changes made to the editable record are applied to the original record after calling the method 'SaveToScriptableObject'. This method can be found on both the editable record and the model of the record.

Trying to edit the original record will result in error messages and your changes not being applied. There is only 1 original record and a maximum of 1 editable copy. Requesting multiple editable copies of a record will result in the same editable record being returned.

Error messages are given when trying to reference a non-prefab to a record. Only Project Folder Objects are allowed in a Record.

Generated Scriptable Objects

Scriptable objects generated by the tool can be viewed but cannot be edited using the default Unity editor. This is to prevent any changes to records that could break the tool. A warning is shown in the inspector of any generated scriptable object, with a button to open the tool instead. All generated scripts are located in Assets/SheetCodes/Resources/ScriptableObjects.

Loading sheets manually

Any sheet is automatically loaded into memory when trying to access it. However it is possible to manage the loading of sheets yourself. The ModelManager has 3 functions to help with loading and unloading of the sheets.

ModelManager.InitializeAll() loads all sheets immediately.

ModelManager.Initialize loads your sheet immediately. This is also called when trying to access a sheet's record that is not loaded yet.

ModelManager.InitializeAsync loads your sheet async. A callback is required when calling the function. It will return true if the sheet is loaded or false if Unload is called on the sheet before Async loading is complete.

ModelManager.Unload unloads the sheet from memory.

Updating from version 1.0, 1.1 to version 1.2

With version 1.2 the modelmanager is no longer a ScriptableObject. To update to this version import the package and **unselect** the following folder when importing.

- SheetCodes/Scripts/GeneratedCode

Open Tools → SheetCodes and make any change that triggers the ModelManager to be regenerated. Recommend to either:

- Change the order of 2 columns in any sheet
- Change the order of 2 rows in any sheet

Press save and wait for the compiler to finish. You will get an error message in your console. Go to SheetCodes/Resources/ScriptableObjects and remove ModelManager. The error message will disappear.

You are now updated to version Sheet Codes 1.2

Updating to version 1.3

With version 1.3 different code needs to be generated for the ModelRecords. To do this first import the package as normal.

After you've done this you need to force a code regeneration. Do this as follows:

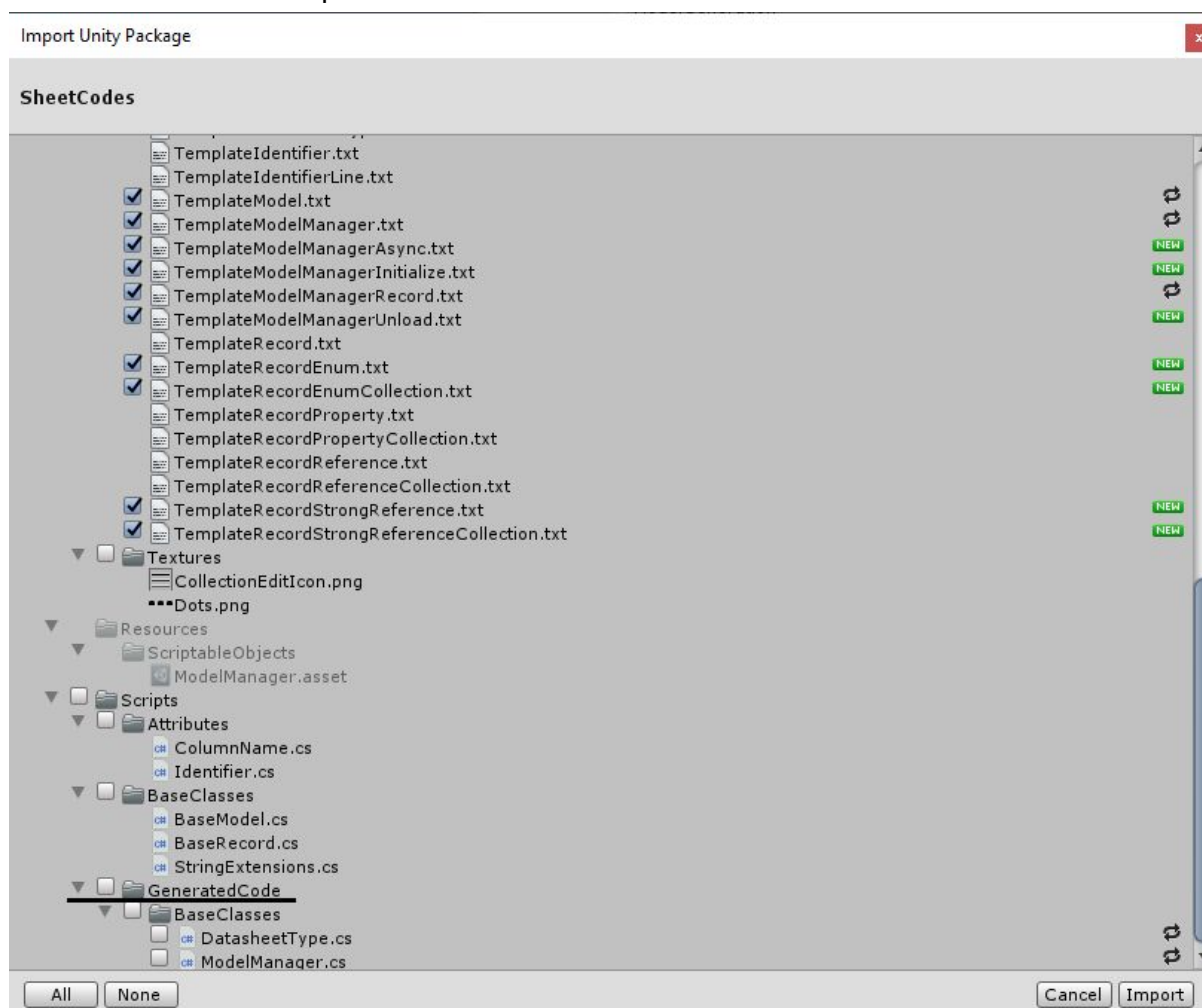
Open Tools → SheetCodes and make any change that triggers the ModelManager to be regenerated. Recommend to either:

- Change the order of 2 columns in any sheet
- Change the order of 2 rows in any sheet

Updating Sheet Codes or Upgrading from Sheet Codes Free

When upgrading to the full version of Sheet Codes or updating Sheet Codes Free makes sure to **unselect** the following folder when importing.

- SheetCodes/Scripts/GeneratedCode



Tool access prevention

The tool cannot operate properly as long as your project contains compiler errors. A prompt will show if an attempt is made to open the tool.

Compiler errors outside Sheet Codes

When editing or removing rows or columns, there is a possibility that the resulting code contains compiler errors. For example, if you have a column called “Health” and you remove this column while still using it in parts of your code, there will be compiler errors. Because the tool cannot create or fill in the scriptable objects until the compiler errors are fixed, a text file containing all data will be stored in Assets/Editor/SheetCodes/SheetCodesJson. After the compiler errors have been fixed, the scriptable objects will be filled in and the text file will be removed.

Compiler errors in Sheet Codes

It is possible that compiler errors occur in the tool’s generated code after removing a class or enum used in one of the columns of Sheet Codes. For example, if we removed the class called “Equipment”, the code generated for “ItemRecord” will contain a compiler error. Each generated property that can throw an exception like this, is surrounded by comments telling you what to remove. In the case the class “Equipment” would no longer exist, we would remove the lines of codes 17 to 20.

```
1  using System;
2  using UnityEngine;
3
4  namespace SheetCodes
5  {
6      [Serializable]
7      public class ItemsRecord : BaseRecord<ItemsIdentifier>
8      {
9          [ColumnName("Damage")] [SerializeField] private int _damage = default;
10         public int Damage { get { return _damage; } }
11
12         //Does this type no longer exist? Delete from here..
13         [ColumnName("Equipment Location")] [SerializeField] private EquipmentSlot _equipmentLocation = default;
14         public EquipmentSlot EquipmentLocation { get { return _equipmentLocation; } }
15         //..To here
16
17         //Does this type no longer exist? Delete from here..
18         [ColumnName("Prefab")] [SerializeField] private Equipment _prefab = default;
19         public Equipment Prefab { get { return _prefab; } }
20         //..To here
21
22         [ColumnName("Crafting Requirements")] [SerializeField] private string[] _craftingRequirements = default;
23         public string[] CraftingRequirements { get { return _craftingRequirements; } }
24         /*PROPERTIES*/
25     }
26 }
27
```

Extra features

The tool comes with many small, somewhat hidden features:

- Scroll horizontally with Ctrl + mouse wheel.
- Scroll vertically with the mouse wheel.
- Left-click and drag the top cell of any column to move it.
- Left-click and drag the first cell of any row to move it.
- Left-click and drag between columns and rows to resize them.
- Left-click and drag the dark line left of the 'Add' button to resize the Array Edit.
- Everything is supported by scrollbars, so there are no limitations to the number of columns, rows, or elements in an Array.

Contact

For questions please email us at: FirstFinal.SheetCodes@gmail.com