Simple Data Tables
User Guide

Overview

Simple Data Tables is a Unity tool which allows game data to be easily stored in structured tables. No coding knowledge is required to create a new table or add/edit its contents. Data tables could be used for a multitude of different purposes, including, loot systems, quest information, inventory storage, game option presets, unlockable skills etc.

Itch.io: https://samcarey.itch.io/simple-data-tables
Download: https://assetstore.unity.com/packages/tools/utilities/simple-data-tables-121783

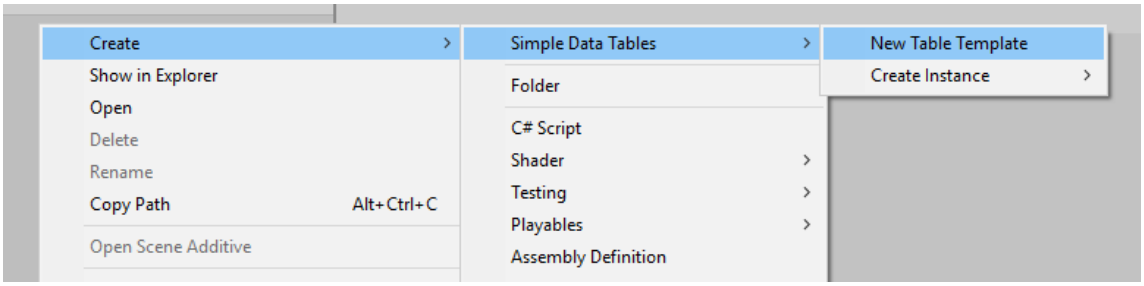
Examples

Examples for this pack can be found under **"SimpleDataTables/Example".**

- The DatatableExampleScene will output some test functions to the console.
- ReadDatatableExample.cs demonstrates many of the included functions.

Creating a New Table Type (Table Templates)

A template is an asset which defines the column types and initial styling options of a data table. A new template can be created by right clicking in the asset view and selecting **“Create/Simple Data Tables/New Table Template”** as seen below.

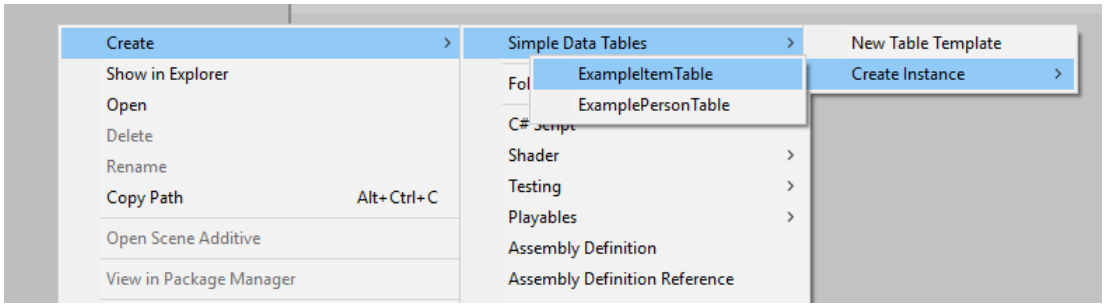


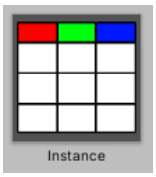
Each attribute must have a name e.g. Age, Height, FavoriteColour etc. Each row will represent one attribute(column) in the final table. All names must follow C# naming rules. The buttons on the right can be used to rearrange or remove an attribute.
Important note: After the table has been created the generate button will need to be clicked again to update the definition with any other changes.

If you are unable to set the widths or colors of each row it means ‘automatic style updating’ is switched on. See Settings page for more information (Page 5).

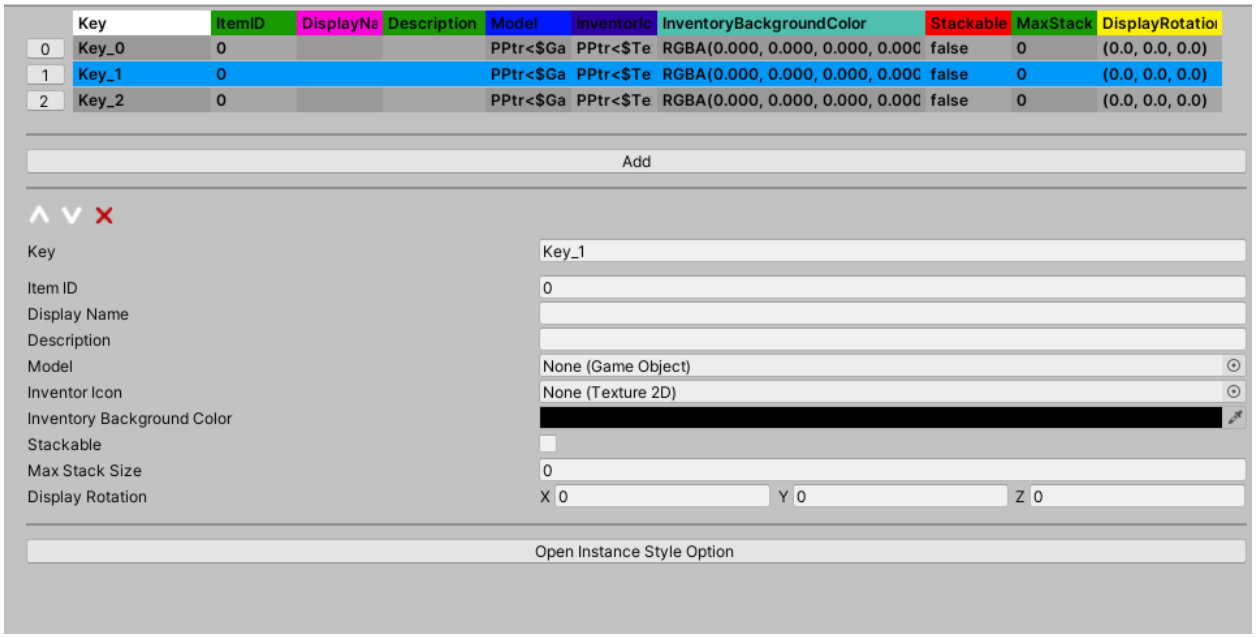
Table Instances

After a table has been generated, an instance can be created. A table instance is the asset used to store table data (rows). You can create an instance in the asset browser by selecting **“Create/Simple Data Tables/Create Instance/Name of Table”**





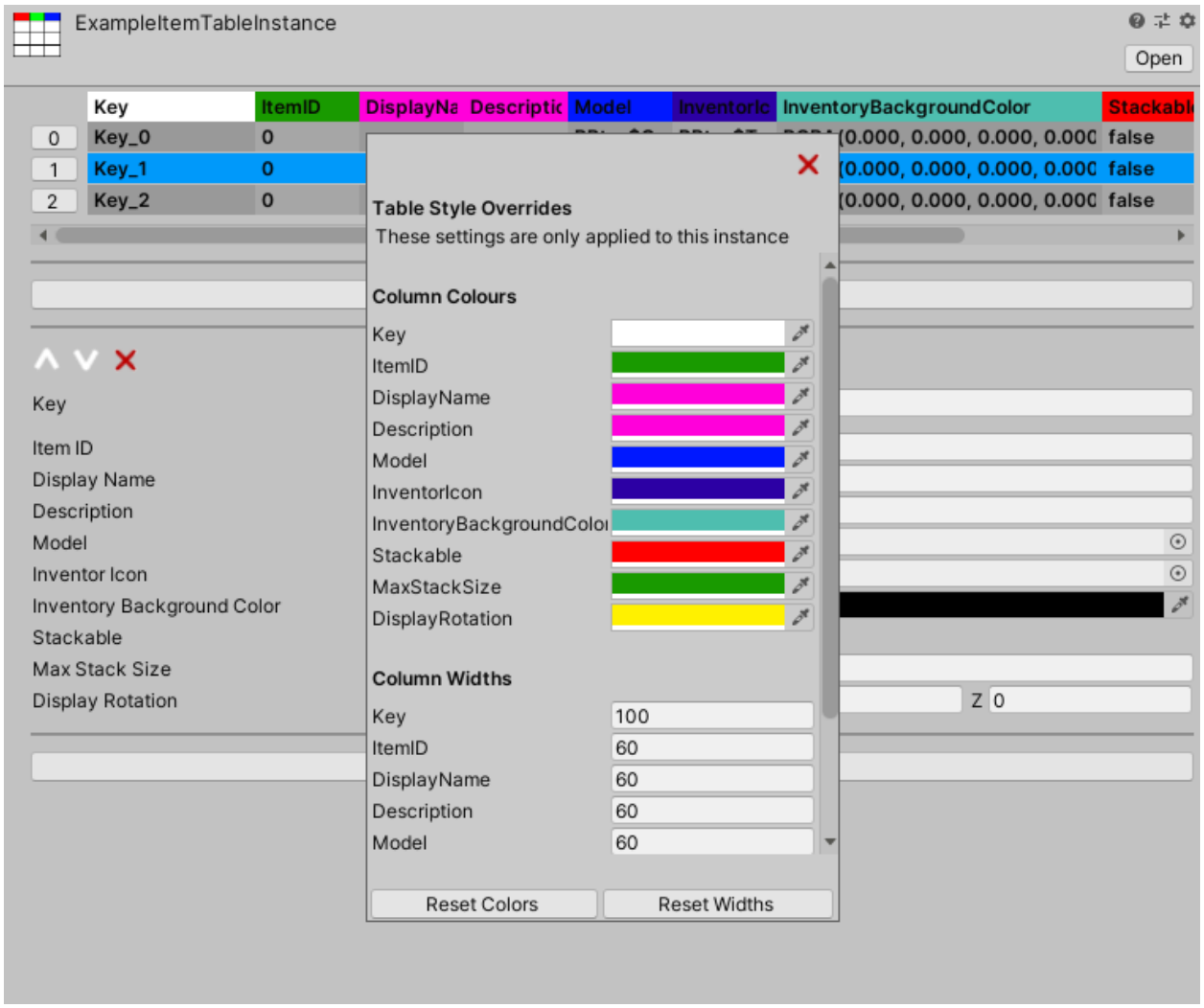
From this page, you can use the ‘add’ button to add a new row. The up, down and remove buttons can be used to rearrange or delete rows.
To make and edit you must first select a row by clicking the index button to the left and all fields should appear below.



Per Instance Styles

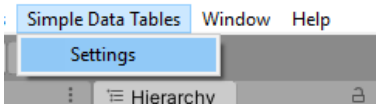
All table instances can have their column colors and widths overridden by clicking the "Open Instance Style Options" button. Any changes here will only affect the current instance of the table.

Note - All of the instance styles will be reset if new columns are added or removed in the template editor.



Settings

There are several general settings which can be customized to your preferences.



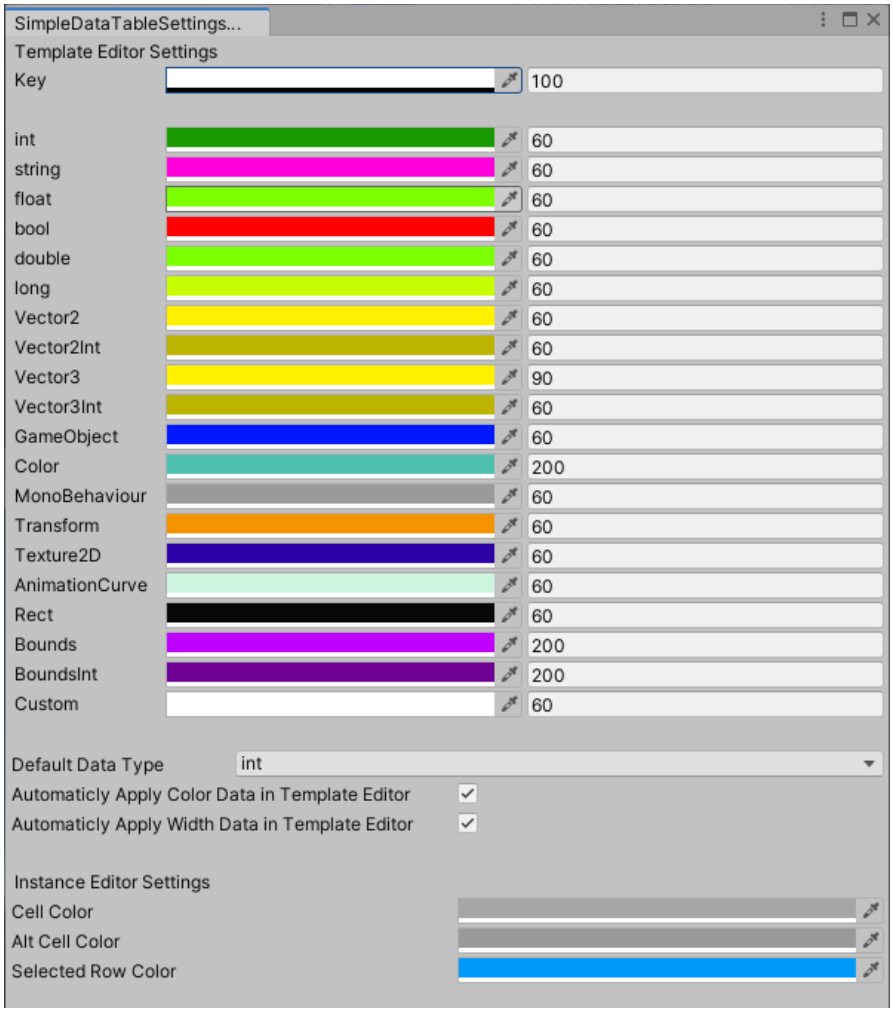
The settings window can be accessed from the top bar under ‘Simple Data Tables’

Template Editor Settings

Default column colors and widths for each available data type can be defined. The default data type to be selected can also be set. If ‘Automatically Apply Color Data’ and ‘Automatically Apply Width Data’ are true, the default colors and widths will be automatically set in the template editor. Keep these false if you wish to customize the colors yourself when creating a template.

Instance Editor Settings

For the instance editor, you can simply adjust some default editor colors.

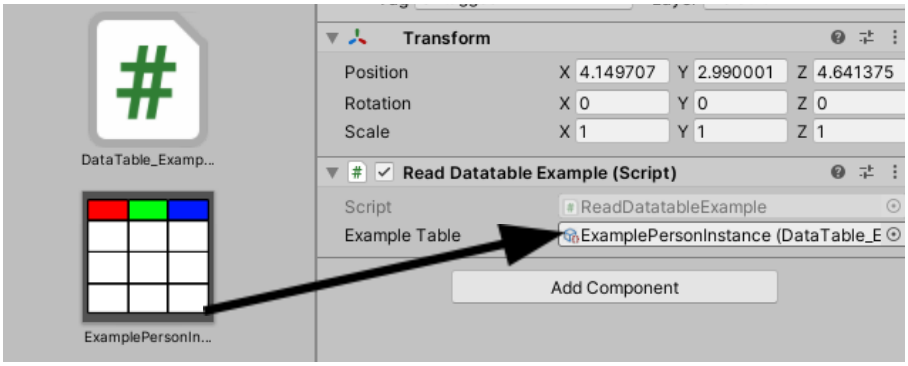


Function Reference

Data Retrieval

To interact with a table in code you must first define a variable to act as a reference to the table you wish to edit.

```
public DataTable ExampleTable;
```



Note on generics (T Variables) - In the case of functions with T you should input the type of row which matches your table. The format for this is always: **DataTableRow_NameOfTable**

public T GetRow<T>(int Index){}
Returns a row from a table at a specific index. Example: **DataTableRow_Example = ExampleTable.GetRow<DataTableRow_Example>(0);**
public T GetRow<T>(string Key){}
Returns the first row in the table which has a specific key. Example: **DataTableRow_Example = ExampleTable.GetRow<DataTableRow_Example>("SomeKey");**

public T[] GetAllRowsWithKey<T>(string Key){}
Returns an array of all rows with a specific key. Example: **DataTableRow_Example[] = ExampleTable.GetRow<DataTableRow_Example>("SomeKey");**

public T[] GetAllRows<T>(){}
Returns an array of all rows.

public T GetRansomRow<T>(){}
Returns a random row from a data table. Example: **DataTableRow_Example = ExampleTable.GetRandomRow<DataTableRow_Example>();**

Helper Functions

public bool HasKey(string Key){}
Returns true if there is at least one row with the passed in key.

public string[] GetAllKeys(){}
Returns a list of all keys present in the table with duplicates included.

public string[] GetAllKeysUnique(){}
Returns a list of all keys present in the table with duplicates removed.

public int Size(){}
Returns the number of rows in a data table.

Edit Table Functions

public void Add(string Key){}
Adds a new row to the bottom of a data table with default values.

public void Remove(string Key, bool MultipleRows){}
Removes a row with a given key from the table. If ‘MultipleRows’ is true all rows with the key will be removed. If ‘MultipleRows’ is false only the first match will be removed.
public void Remove(string Key){}
Removes a row from a table at a given index.

Quick Function Reference

Data Retrieval

```
public T GetRow<T>(int Index){}
public T GetRow<T>(string Key){}
public T[] GetAllRowsWithKey<T>(string Key){}
public T[] GetAllRows<T>(){}
public T GetRansomRow<T>(){}
```

Helper Functions

```
public bool HasKey(string Key){}
public string[] GetAllKeys(){}
public string[] GetAllKeysUnique(){}
public int Size(){}
```

Edit Table Functions

```
public void Add(string Key){}
public void Remove(string Key, bool MultipleRows){}
public void Remove(string Key){}
```