Table of Contents

# [Serializable2DArray<T>](#_Serializable2DArray<T>_Documentatio)

* **Overview**
* **Creating a Serializable 2D Array**
* **Accessing and Modifying Values**
* **Serialization in Unity**
* **Utility Methods**
  + **Find**
  + **Exists**
  + **IndexOf**
  + **GetLength**
* **Equality and Comparisons**
* **Example Use Case in Unity**

# [SerializableDictionary<TKey, TValue>](#_SerializableDictionary<TKey,_TValue)

* **Purpose and Overview**
* **Creating a Serializable Dictionary**
* **Accessing and Modifying Entries**
* **Inspector Serialization**
* **Enumerating and Copying**
* **Implicit and Explicit Conversion**
* **Example Use Case in Unity**
* **Notes and Limitations**

# [SerializableDateTime](#_SerializableDateTime_Documentation)

* **Class Overview**
* **Fields and Properties**
* **Read-Only Date and Time Components**
* **Constructors**
* **Methods**
* **Equality and Comparison Operators**
* **Implicit Conversion**
* **Usage Example in Unity**

# [SerializableHashSet<T>](#_SerializableHashSet<T>_Documentatio)

* **Class Overview**
* **Constructors**
* **Access and Modification**
  + **Indexer**
  + **Adding**
  + **Removing**
* **Query and Utility Methods**
* **Properties**
* **Enumeration and Copying**
* **Casting Operators**
* **Example Usage in Unity**

# [SerializableQueue<T>](#_SerializableQueue<T>_Documentation)

* **Class Overview**
* **Constructors**
* **Access and Properties**
  + **Indexer**
  + **Count and Empty Check**
* **Queue Operations**
  + **Enqueue**
  + **Dequeue**
  + **Peek**
* **Modification Methods**
* **Enumeration and Conversion**
* **Sorting**
* **Serialization Handling**
* **Implicit and Explicit Casting**
* **Example Usage in Unity**

# Serializable2DArray<T> Documentation

Serializable2DArray<T> is a generic, Unity-serializable two-dimensional array wrapper. It allows you to store and serialize 2D grid-like data in the Unity Inspector while still maintaining easy 2D array indexing and common utility methods.

**Creating a Serializable 2D Array**

To initialize the array, provide the width and height:



This creates a 5 (x) by 3 (y) grid backed by a flat 1D array.

**Accessing and Modifying Values**

You can treat it like a 2D array using the indexer:

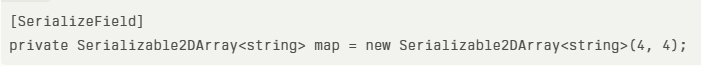


If you try to access values outside the boundaries, it will throw an IndexOutOfRangeException.

**Serialization in Unity**

The class is marked as [Serializable] and uses [SerializeField], so it can appear in the Unity Inspector (if used as part of a MonoBehaviour or ScriptableObject).  
The built-in OnAfterDeserialize ensures the array is always re-created properly after Unity’s serialization.

Example:



**Utility Methods**

**Find**

Searches through the grid and returns the first element matching a predicate:



If no match is found, it returns default(T) (null for reference types, 0 for int, etc.).

**Exists**

Checks if at least one element matches a given condition:



DReturns true if found, otherwise false.

**IndexOf**

Finds the first occurrence of a given value and returns its grid position as a Vector2Int:

A close-up of a computer screen

AI-generated content may be incorrect.

If not found, it returns (-1, -1).

**GetLength**

Returns the dimensions of the array:

A close up of text

AI-generated content may be incorrect.

**Equality and Comparisons**

Two Serializable2DArray<T> objects can be compared using == and !=. The comparison checks:

* Same dimensions (width and height)
* Same values stored in data

Example:

A white background with black text

AI-generated content may be incorrect.

**Example Use Case in Unity**

Here’s an example of using a Serializable2DArray to manage a tile-based map:

A screenshot of a computer code

AI-generated content may be incorrect.

This setup allows you to easily edit and inspect the 2D grid in Unity’s Inspector.

# 

# SerializableDateTime Documentation

SerializableDateTime is a Unity-serializable wrapper around the standard System.DateTime struct.  
It enables safe serialization and editing of date and time information within the Unity Inspector by breaking the date and time into separate fields with validation.

**Class Overview**

This class exposes the date and time as individual integer fields (year, month, day, hour, minute, and second) that Unity can serialize and enforce ranges through inspector sliders.  
It manages conversion to and from DateTime internally, handling invalid dates with safe fallbacks.

**Fields and Properties**

* year (int, 1 to 9999): Year component of the date.
* month (int, 1 to 12): Month component.
* day (int, 1 to 31): Day component.
* hour (int, 0 to 23): Hour component in 24-hour format.
* minute (int, 0 to 59): Minute component.
* second (int, 0 to 59): Second component.

You can get or set the whole DateTime via the Value property:

A white background with black text

AI-generated content may be incorrect.

**Read-Only Date and Time Components**

The object exposes read-only properties reflecting the DateTime values derived from the current state:

* Year, Month, Day
* Hour, Minute, Second
* DayOfWeek, DayOfYear
* Millisecond, Ticks
* TimeOfDay (TimeSpan)

**Constructors**

* Default constructor: Initializes to default values (year 1900, month 1, etc.).
* SerializableDateTime(DateTime dateTime): Initialize from existing DateTime.
* Detailed constructor with components:



All inputs clamp to valid ranges.

**Methods**

* AddSeconds(double seconds): Adds seconds to the current date/time and returns the updated object.
* ToString(): Returns the date/time formatted as "MM-dd-yyyy HH:mm:ss".

Example:



**Equality and Comparison Operators**

* Supports comparison operators (==, !=, <, >, <=, >=) based on the underlying DateTime value.
* Implements Equals and GetHashCode accordingly.

Example:

A white background with green text

AI-generated content may be incorrect.

**Implicit Conversion**

Convenient implicit conversions allow interchange between SerializableDateTime and DateTime:

A black text on a white background

AI-generated content may be incorrect.

**Usage Example in Unity**

A screen shot of a computer code

AI-generated content may be incorrect.

This allows editing and tracking of dates and times directly via the Unity Inspector with validation on each field.

# SerializableDictionary<TKey, TValue> Documentation

SerializableDictionary<TKey, TValue> is a generic class that provides a Unity-serializable dictionary. It enables using dictionaries as serialized fields in the Unity Inspector, ensuring reliable storage and retrieval of key/value pairs during Play Mode and asset serialization.

**Purpose and Overview**

Unity’s built-in serialization system does not support generic dictionaries directly. SerializableDictionary<TKey, TValue> uses a serializable list of key-value pairs for Inspector support. It synchronizes a standard C# dictionary behind the scenes and offers the full functionality of IDictionary<TKey, TValue> within your scripts.

**Creating a Serializable Dictionary**

To declare and create a serializable dictionary, specify the types for the key and value:

Initialization example:

A close-up of a word

AI-generated content may be incorrect.

**Accessing and Modifying Entries**

You can use most dictionary features:

To add, remove, or check for keys:

A screenshot of a computer code

AI-generated content may be incorrect.

**Inspector Serialization**

When the dictionary is serialized by Unity (in Editor or in builds), the key-value data is stored in an internal list.  
Deserialization automatically rebuilds the dictionary, ensuring data integrity and unique keys.

Note: Keys must be unique and must not be null or Unity will skip the entry during deserialization.

**Enumerating and Copying**

You can enumerate keys and values:

A screenshot of a computer code

AI-generated content may be incorrect.

To copy entries to an array:

D

**Implicit and Explicit Conversion**

* Implicit conversion to Dictionary<TKey, TValue>:  
  You can get the internal dictionary directly:



* Explicit conversion from Dictionary<TKey, TValue>:  
  You can create a serializable version from any dictionary:



**Example Use Case in Unity**

Below is an example for tracking scores with Unity serialization:

A screen shot of a computer code

AI-generated content may be incorrect.

This allows you to inspect and modify the scores in the Unity Inspector.

**Notes and Limitations**

* All key and value types used must themselves be serializable by Unity.
* Complex or custom key types may require implementing Equals and GetHashCode properly.
* Serialization callbacks keep the internal dictionary and serialized list in sync automatically.

# SerializableHashSet<T> Documentation

SerializableHashSet<T> is a Unity-serializable collection class that mimics the behavior of a HashSet<T> but stores items in a list for compatibility with Unity serialization.  
It ensures uniqueness of items while providing familiar set operations and Unity Inspector support.

**Class Overview**

Unlike the standard HashSet<T>, Unity cannot serialize hash sets directly.  
This class stores elements internally as a List<T>, enforcing uniqueness when adding or modifying elements.  
It implements IEnumerable<T> and ICollection<T> interfaces, providing collection functionality.

**Constructors**

* Default constructor: Creates an empty set.



* Collection constructor: Initializes the set with unique elements from an IEnumerable<T>.



**Access and Modification**

**Indexer**

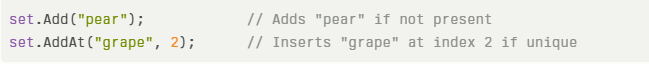
Access or modify elements by index:



Modifying with an existing value will throw an exception to maintain uniqueness.

**Adding**

* Add(T item): Adds the item if not already present, returning true.
* AddAt(T item, int index): Inserts an item at a specific index if unique.

**Removing**

* Remove(T item): Removes the item if it exists, returning true.
* RemoveAll(Predicate<T> match): Removes all items matching a condition and returns the count of removed items.



**Query and Utility Methods**

* Contains(T item): Checks if an item exists.
* Find(Func<T, bool> predicate): Returns the first item matching the predicate or default if none found.
* Clear(): Empties the set.
* ToArray(): Returns an array copy of the items.

**Properties**

* Items: Read-only list of the current items.
* Count: Number of items in the set.
* IsReadOnly: Always false.

**Enumeration and Copying**

Supports iteration via foreach loops, and copying contents into an array:

A screenshot of a computer code

AI-generated content may be incorrect.

**Casting Operators**

* Implicit conversion to HashSet<T>: Converts this serializable set to a standard HashSet<T>.



* Explicit conversion from HashSet<T>: Creates a serializable set from a standard hash set.



**Example Usage in Unity**

A screen shot of a computer code

AI-generated content may be incorrect.This setup allows unique item tracking with a serializable collection visible and editable in the Inspector.

# SerializableQueue<T> Documentation

SerializableQueue<T> is a Unity-serializable queue collection that maintains insertion order and supports serialization in the Unity Inspector.  
It preserves the typical behavior of a Queue<T> while using an internal list for Unity-compatible serialization.

**Class Overview**

This class combines a Unity-serializable List<T> with a runtime Queue<T>.  
On deserialization, the queue is rebuilt from the list to maintain data integrity.  
All queue operations update both the list and the queue to keep them in sync.

**Constructors**

* Default constructor: Creates an empty queue.



* Collection constructor: Initializes the queue with elements from any enumerable collection.



**Access and Properties**

**Indexer**

Access elements by index (read-only):



Throws an exception if the index is out of range.

**Count and Empty Check**

* Count: Number of elements in the queue.
* IsEmptyOrNull: True if the queue is null or contains no elements.

**Queue Operations**

**Enqueue**

Add single or multiple items to the queue:

A close-up of a computer code

AI-generated content may be incorrect.

**Dequeue**

Remove and return the item at the front of the queue:



Returns default value if queue is empty.

**Peek**

* PeekFirst(): Returns the front item without removing it.
* PeekLast(): Returns the last item inserted without removing it.

Returns default if the queue is empty.

**Modification Methods**

* Remove(T item): Removes the first occurrence of an item from the queue.
* RemoveAll(Predicate<T> match): Removes all items matching a predicate and returns how many were removed.
* Clear(): Removes all items.

**Enumeration and Conversion**

* Implements IEnumerable<T> to allow iteration:

A close-up of a white background

AI-generated content may be incorrect.

* ToArray(): Gets an array snapshot of the queue contents.

**Sorting**

* SortBy<TKey>(Func<T, TKey> keySelector, bool ascending = true):  
  Sorts the queue elements based on a key selector, either ascending or descending.

Example:



**Serialization Handling**

* OnAfterDeserialize(): Called by Unity after deserialization to rebuild the internal queue from the serialized list.
* SyncQueueFromList(): Internal helper to synchronize the queue with the list.

**Implicit and Explicit Casting**

* Implicit conversion to native Queue<T>:



* Explicit conversion from native Queue<T>:



**Example Usage in Unity**

A computer screen shot of a program code

AI-generated content may be incorrect.

This enables serialized, ordered task management visible and editable in the Unity Inspector.