C# Extensions for Unity

Overview

This document provides a detailed overview of various C# extension methods available for use in Unity. The extensions cover components, dictionaries, floats, game objects, integers, lists, MonoBehaviours, objects, rigid bodies, strings, transforms, vectors, and more. Each section includes method signatures, descriptions, and usage examples.

ComponentExtensions

`AddComponent<T>(this Component component)`

Description:
Adds a component of type `T` to the GameObject of the given `Component`.

Parameters:

- `component`: The component to which the new component will be added.

Returns:

- The added component of type `T`.

Usage Example:

```
var rigidbody = playerMovement.AddComponent<Rigidbody>();
```

`GetOrAddComponent<T>(this Component component)`

Description:
Gets an existing component of type `T` or adds it if it doesn't exist.

Parameters:

- `component`: The component from which to get or add the new component.

Returns:

- The existing or newly added component of type `T`.

Usage Example:

```
var newComponent = myComponent.GetOrAddComponent<NewComponent>();
```

`HasComponent<T>(this Component component)`

Description:
Checks if the component has a component of type `T`.

Parameters:

- `component`: The component to check.

Returns:

- `true` if the component has a component of type `T`, otherwise `false`.

Usage Example:

```
bool hasMyComponent = myComponent.HasComponent<MyComponent>();
```

DecimalExtensions

`TruncateTo(this decimal n, uint digits)`

Description:
Truncates a decimal number to a specified number of decimal places.

Parameters:

- `n`: The decimal number to truncate.
- `digits`: The number of decimal places.

Returns:

- The truncated decimal number.

Usage Example:

```
decimal truncatedValue = 3.14159m.TruncateTo(2);  // 3.14
```

DictionaryExtensions

`AddOrUpdate<TKey, TValue>(this Dictionary<TKey, TValue> dictionary, TKey key, TValue value)`

Description:
Adds a new key-value pair or updates the value for an existing key.

Parameters:

- `dictionary`: The dictionary to modify.
- `key`: The key to add or update.
- `value`: The value associated with the key.

Usage Example:

```
myDictionary.AddOrUpdate("key", "value");
```

FloatExtensions

`RandomBias(this float value, float range, bool useNegativeBias = true)`

Description:
Applies a random bias to a float value within a specified range.

Parameters:

- `value`: The base value.
- `range`: The range for the random bias.
- `useNegativeBias`: Whether to include negative bias.

Returns:

- The biased float value.

Usage Example:

```
float newSpeed = speed.RandomBias(0.1f);
```

`Round(this float value, int digits = 0)`

Description:
Rounds a float to a specified number of decimal places.

Parameters:

- `value`: The float to round.
- `digits`: The number of decimal places.

Returns:

- The rounded float value.

Usage Example:

```
float roundedValue = 3.14159f.Round(2);  // 3.14
```

`IsApproximatelyEqual(this float value, float other,float tolerance = 0.0001f)`

Description:
Checks if the given float is approximately equal to another float in a given tolerance.

`ToPercentage(this float value, float total)`

Description:
Convert the value to a percentage

GameObjectExtensions

`GetOrAddComponent<T>(this GameObject gameObject)`

Description:
Gets an existing component of type `T` or adds it if it doesn't exist.

Parameters:

- `gameObject`: The GameObject to modify.

Returns:

- The existing or newly added component of type `T`.

Usage Example:

```
var myComponent = myGameObject.GetOrAddComponent<MyComponent>();
```

ToggleActive(this GameObject gameObject)

Description:
Toggles the active state of the GameObject.

Parameters:

- `gameObject`: The GameObject to toggle.

Usage Example:

```
myGameObject.ToggleActive();
```

HasComponent<T>(this GameObject gameObject)

Description:
Checks if the game object have the given component

DestroyAllChildren(this GameObject gameObject)

Description:
Destroy all child game objects

AddComponentIfMissing<T>(this GameObject gameObject)

Description:
Add given component if its missing

IntExtensions

`ToAbbreviatedString(this int n, uint digits = 0)`

Description:
Converts an integer to an abbreviated string (e.g., "1k", "2m").

Parameters:

- `n`: The integer to convert.
- `digits`: The number of decimal places for the abbreviation.

Returns:

- The abbreviated string.

Usage Example:

```
string abbreviated = 1500.ToAbbreviatedString();  // "1.5k"
```

`RoundToMultipleOf(this int n, int binSize)`

Description:
Rond the given number to a multiple of another number

ListExtensions

`GetRandomItem<T>(this IList<T> list)`

Description:
Gets a random item from the list.

Parameters:

- `list`: The list to get the random item from.

Returns:

- A random item from the list.

Usage Example:

```
var randomItem = myList.GetRandomItem();
```

`RemoveLastItem<T>(this IList<T> source, int n = 1)`

Description:
Removes last Item.

`RemoveNulls<T>(this List<T> list)`

Description:
Removes all Null values from the list.

MonoBehaviourExtensions

`DelayedExecution(this MonoBehaviour monoBehaviour, float delay, Action callback)`

Description:
Executes a callback after a specified delay.

Parameters:

- `monoBehaviour`: The MonoBehaviour to use for starting the coroutine.
- `delay`: The delay before executing the callback.
- `callback`: The action to execute.

Usage Example:

```
this.DelayedExecution(2f, () => Debug.Log("Delayed action executed."));
```

`AddComponentIfMissing<T>(this MonoBehaviour behaviour)`

Description:
Add given component if its missing

`GetOrAddComponent<T>(this MonoBehaviour behaviour)`

Description:
Get the component if it exists. Else get the component after adding it

ObjectExtensions

`DestroyGameObject(this Object value)`

Description:
Destroys a GameObject or MonoBehaviour.

Parameters:

- `value`: The object to destroy.

Usage Example:

```
myObject.DestroyGameObject();
```

RigidbodyExtensions

`ChangeDirection(this Rigidbody rigidbody, Vector3 direction)`

Description:
Changes the direction of a Rigidbody without changing its speed.

Parameters:

- `rigidbody`: The Rigidbody to modify.
- `direction`: The new direction.

Usage Example:

```
myRigidbody.ChangeDirection(Vector3.forward);
```

StringExtensions

ToEnum<T>

Description: Converts the given string to an enum of type T. If the string cannot be converted, an exception is thrown.

Usage:

```
MyEnum myEnumValue = "ValueName".ToEnum<MyEnum>();
```

- **Parameters**: None
- **Returns**: Enum value of type T.
- **Throws**: ArgumentException if the conversion fails.

Example:

```
// Convert "Monday" to a DayOfWeek enum
DayOfWeek day = "Monday".ToEnum<DayOfWeek>();
```

Truncate

Description: Truncates the string to the specified maximum length. If the string is longer than the maximum length, it cuts off the extra characters.

Usage:

```
string shortString = longString.Truncate(10);
```

- **Parameters**: maxLength (int) - Maximum length of the truncated string.
- **Returns**: The truncated string.

Example:

```
// Truncate to 5 characters
```

```
string result = "HelloWorld".Truncate(5); // Result: "Hello"
```

ToTitleCase

Description: Converts the string to title case (capitalizing the first letter of each word).

```
string titleCase = "hello world".ToTitleCase();
```

- **Parameters**: None
- **Returns**: The string in title case.

```
// Convert to title case
string result = "hello world".ToTitleCase(); // Result: "Hello World"
```

IsNullOrEmpty

Description: Checks if the string is null or empty.

```
bool isEmpty = myString.IsNullOrEmpty();
```

- **Parameters**: None
- **Returns**: true if the string is null or empty, false otherwise.

```
// Check if the string is null or empty
bool result = "".IsNullOrEmpty(); // Result: true
```

IsNullOrWhiteSpace

Description: Checks if the string is null, empty, or consists only of white-space characters.

Usage:

```
bool isWhiteSpace = myString.IsNullOrWhiteSpace();
```

- **Parameters**: None
- **Returns**: true if the string is null, empty, or contains only white-space characters, false otherwise.

Example:

```
// Check if the string is null, empty, or white space
bool result = "   ".IsNullOrWhiteSpace(); // Result: true
```

Reverse

Description: Reverses the characters in the string.

```
string reversed = myString.Reverse();
```

- **Parameters**: None
- **Returns**: The string with characters in reverse order.

Example:

```
// Reverse the string
string result = "abc".Reverse(); // Result: "cba"
```

RemoveWhitespace

Description: Removes all white-space characters from the string.

```
string noWhitespace = myString.RemoveWhitespace();
```

- **Parameters**: None
- **Returns**: The string with all white-space characters removed.

Example:

```
// Remove whitespace from the string
string result = "a b c".RemoveWhitespace(); // Result: "abc"
```

ToCamelCase

Description: Converts the string to camel case (lowercase first letter and the rest of the string unchanged).

Usage:

```
string camelCase = myString.ToCamelCase();
```

- **Parameters**: None
- **Returns**: The string converted to camel case.

Example:

```
// Convert to camel case
string result = "HelloWorld".ToCamelCase(); // Result: "helloWorld"
```

SplitCamelCase

Description: Splits a camel case string into separate words with spaces in between.

Usage:

```
string splitString = myString.SplitCamelCase();
```

- **Parameters**: None
- **Returns**: The string with camel case split into words.

Example:

```
// Split camel case into words
string result = "HelloWorld".SplitCamelCase(); // Result: "Hello
World"
```

TransformExtensions

SetLocalPosition

Description: Sets the local position of a Transform. You can specify new values for x, y, and z coordinates individually. If a coordinate is not specified, it remains unchanged.

Usage:

```
transform.SetLocalPosition(x: 1f, y: 2f);
```

- **Parameters**:
  - x (float?) - Optional new x-coordinate.
  - y (float?) - Optional new y-coordinate.
  - z (float?) - Optional new z-coordinate.
- **Returns**: Void

Example:

```
// Set the local position to (3, 5, 7), keep existing values for unspecified coordinates
transform.SetLocalPosition(x: 3, y: 5, z: 7);
```

SetWorldPosition

Description: Sets the world position of a Transform. You can specify new values for x, y, and z coordinates individually. If a coordinate is not specified, it remains unchanged.

Usage:

```
transform.SetWorldPosition(y: 10f);
```

- **Parameters**:
  - x (float?) - Optional new x-coordinate.
  - y (float?) - Optional new y-coordinate.
  - z (float?) - Optional new z-coordinate.
- **Returns**: Void

Example:

```
// Set the world position to (0, 10, 0), keep existing values for unspecified coordinates
transform.SetWorldPosition(y: 10);
```

SetPositionX

Description: Sets the x-coordinate of the Transform's world position while keeping the current values for y and z coordinates.

Usage:

```
transform.SetPositionX(7f);
```

- **Parameters**:
  - x (float) - New x-coordinate.
- **Returns**: Void

Example:

```
// Set the x-coordinate to 7, keep existing values for y and z
transform.SetPositionX(7);
```

SetPositionY

Description: Sets the y-coordinate of the Transform's world position while keeping the current values for x and z coordinates.

Usage:

```
transform.SetPositionY(3f);
```

- **Parameters**:
  - y (float) - New y-coordinate.
- **Returns**: Void

Example:

```
// Set the y-coordinate to 3, keep existing values for x and z
```

```
transform.SetPositionY(3);
```

SetPositionZ

Description: Sets the z-coordinate of the Transform's world position while keeping the current values for x and y coordinates.

Usage:

```
transform.SetPositionZ(-5f);
```

- **Parameters**:
  - z (float) - New z-coordinate.
- **Returns**: Void

Example:

```
// Set the z-coordinate to -5, keep existing values for x and y
transform.SetPositionZ(-5);
```

SetLocalEulerAngles

Description: Sets the local rotation angles (Euler angles) of a Transform. You can specify new values for x, y, and z angles individually. If an angle is not specified, it remains unchanged.

Usage:

```
transform.SetLocalEulerAngles(x: 30f, y: 60f);
```

- **Parameters**:
  - x (float?) - Optional new x-angle.
  - y (float?) - Optional new y-angle.
  - z (float?) - Optional new z-angle.
- **Returns**: Void

Example:

```
// Set local Euler angles to (30, 60, 90), keep existing values for unspecified angles
transform.SetLocalEulerAngles(x: 30, y: 60, z: 90);
```

SetWorldEulerAngles

Description: Sets the world rotation angles (Euler angles) of a Transform. You can specify new values for x, y, and z angles individually. If an angle is not specified, it remains unchanged.

Usage:

```
transform.SetWorldEulerAngles(z: 90f);
```

- **Parameters**:
  - x (float?) - Optional new x-angle.
  - y (float?) - Optional new y-angle.
  - z (float?) - Optional new z-angle.
- **Returns**: Void

Example:

```
// Set world Euler angles to (0, 0, 90), keep existing values for unspecified angles
transform.SetWorldEulerAngles(z: 90);
```

Vector2Extensions

WithX

Description: This extension method creates a new Vector2 where only the x-coordinate is changed to a specified value, while keeping the y-coordinate from the original vector unchanged.

Usage:

```
Vector2 original = new Vector2(3, 5);
```

```
Vector2 modified = original.WithX(10);
```

- **Parameters**:
  - x (float) - The new x-coordinate value for the vector.
- **Returns**: Vector2 - A new Vector2 instance with the specified x-coordinate and the original y-coordinate.

Example:

```
// Given a vector (3, 5), changing x to 10
Vector2 original = new Vector2(3, 5);
Vector2 modified = original.WithX(10);
// modified is now (10, 5)
```

WithY

Description: This extension method creates a new Vector2 where only the y-coordinate is changed to a specified value, while keeping the x-coordinate from the original vector unchanged.

Usage:

```
Vector2 original = new Vector2(3, 5);
Vector2 modified = original.WithY(20);
```

- **Parameters**:
  - y (float) - The new y-coordinate value for the vector.
- **Returns**: Vector2 - A new Vector2 instance with the specified y-coordinate and the original x-coordinate.

Example:

```
// Given a vector (3, 5), changing y to 20
Vector2 original = new Vector2(3, 5);
Vector2 modified = original.WithY(20);
// modified is now (3, 20)
```

SetMagnitude

Description: This extension method adjusts the magnitude (length) of the Vector2 to a specified value while keeping its direction. It normalizes the vector and then scales it to the new magnitude.

Usage:

```
Vector2 original = new Vector2(3, 4);
Vector2 modified = original.SetMagnitude(10);
```

- **Parameters**:
  - magnitude (float) - The new magnitude (length) for the vector.
- **Returns**: Vector2 - A new Vector2 instance with the specified magnitude and the same direction as the original vector.

Example:

```
// Given a vector (3, 4) with magnitude 5, set its magnitude to 10
Vector2 original = new Vector2(3, 4);
// Original magnitude is sqrt(3^2 + 4^2) = 5
Vector2 modified = original.SetMagnitude(10);
// modified is now (6, 8), with magnitude 10
```

Vector3Extensions

WithX

Description: Creates a new `Vector3` where only the x-coordinate is changed to a specified value, while keeping the y and z coordinates from the original vector unchanged.

Usage:

```
Vector3 original = new Vector3(1, 2, 3);
```

```
Vector3 modified = original.WithX(10);
```

- **Parameters**:
  - `x` (float) - The new x-coordinate value for the vector.
- **Returns**: `Vector3` - A new `Vector3` instance with the specified x-coordinate and the original y and z coordinates.

Example:

```
// Given a vector (1, 2, 3), changing x to 10
Vector3 original = new Vector3(1, 2, 3);
Vector3 modified = original.WithX(10);
// modified is now (10, 2, 3)
```

`WithY`

Description: Creates a new `Vector3` where only the y-coordinate is changed to a specified value, while keeping the x and z coordinates from the original vector unchanged.

Usage:

```
Vector3 original = new Vector3(1, 2, 3);
Vector3 modified = original.WithY(20);
```

- **Parameters**:
  - `y` (float) - The new y-coordinate value for the vector.
- **Returns**: `Vector3` - A new `Vector3` instance with the specified y-coordinate and the original x and z coordinates.

Example:

```
// Given a vector (1, 2, 3), changing y to 20
Vector3 original = new Vector3(1, 2, 3);
Vector3 modified = original.WithY(20);
// modified is now (1, 20, 3)
```

Overloads:

1. **WithY (Transform target)**:
    - **Description**: Creates a new `Vector3` where the y-coordinate is set to the y-coordinate of a specified `Transform`'s position, while keeping the x and z coordinates from the original vector.

Usage:

```
Vector3 original = new Vector3(1, 2, 3);
Transform target = someTransform;
Vector3 modified = original.WithY(target);
```

    -
    - **Parameters**:
        - `target` (Transform) - The `Transform` whose y-coordinate will be used.
    - **Returns**: `Vector3` - A new `Vector3` instance with the y-coordinate from the `Transform` and the original x and z coordinates.
2. **WithY (Vector3 target)**:
    - **Description**: Creates a new `Vector3` where the y-coordinate is set to the y-coordinate of a specified `Vector3`, while keeping the x and z coordinates from the original vector.

Usage:

```
Vector3 original = new Vector3(1, 2, 3);
Vector3 target = new Vector3(4, 5, 6);
Vector3 modified = original.WithY(target);
```

    -
    - **Parameters**:
        - `target` (Vector3) - The `Vector3` whose y-coordinate will be used.
    - **Returns**: `Vector3` - A new `Vector3` instance with the y-coordinate from the `Vector3` and the original x and z coordinates.

Examples:

```
// Changing y to the y-coordinate of a Transform's position
Vector3 original = new Vector3(1, 2, 3);
Transform target = someTransform;
Vector3 modified = original.WithY(target);
// modified is now (1, target.position.y, 3)
```

```
// Changing y to the y-coordinate of another Vector3
Vector3 original = new Vector3(1, 2, 3);
Vector3 target = new Vector3(4, 5, 6);
Vector3 modified = original.WithY(target);
// modified is now (1, 5, 3)
```

`WithZ`

Description: Creates a new `Vector3` where only the z-coordinate is changed to a specified value, while keeping the x and y coordinates from the original vector unchanged.

Usage:

```
Vector3 original = new Vector3(1, 2, 3);
Vector3 modified = original.WithZ(30);
```

- **Parameters**:
  - `z` (float) - The new z-coordinate value for the vector.
- **Returns**: `Vector3` - A new `Vector3` instance with the specified z-coordinate and the original x and y coordinates.

Example:

```
// Given a vector (1, 2, 3), changing z to 30
Vector3 original = new Vector3(1, 2, 3);
Vector3 modified = original.WithZ(30);
// modified is now (1, 2, 30)
```

Overloads:

1. **WithZ (Transform target)**:
   - **Description**: Creates a new `Vector3` where the z-coordinate is set to the z-coordinate of a specified `Transform`'s position, while keeping the x and y coordinates from the original vector.

Usage:

```
Vector3 original = new Vector3(1, 2, 3);
Transform target = someTransform;
```

```
Vector3 modified = original.WithZ(target);
```

    -
    - **Parameters**:
        - `target` (Transform) - The `Transform` whose z-coordinate will be used.
    - **Returns**: `Vector3` - A new `Vector3` instance with the z-coordinate from the `Transform` and the original x and y coordinates.

Examples

```
// Changing z to the z-coordinate of a Transform's position
Vector3 original = new Vector3(1, 2, 3);
Transform target = someTransform;
Vector3 modified = original.WithZ(target);
// modified is now (1, 2, target.position.z)
```

`WithRandomBias`

Description: Adds a random bias to each component of the `Vector3`. This can be useful for adding variability or randomness to positions, velocities, or other vector properties.

Usage:

```
Vector3 original = new Vector3(1, 2, 3);
Vector3 modified = original.WithRandomBiase(0.5f);
```

- **Parameters**:
    - `biasValue` (float) - The maximum amount of random bias to add or subtract.
- **Returns**: `Vector3` - A new `Vector3` with each component randomly biased by the given value.

Example:

```
// Adding a random bias with a maximum of 0.5 to each component
Vector3 original = new Vector3(1, 2, 3);
Vector3 modified = original.WithRandomBiase(0.5f);
// modified might be (1.2, 2.3, 2.8), with random variations in each
component
```

Overload:

1. **WithRandomBiase (Vector3 biasValue)**:
    - **Description**: Adds a random bias to each component of the `Vector3` based on individual bias values for x, y, and z. This allows for different levels of randomness for each component.

Usage:
```
Vector3 original = new Vector3(1, 2, 3);
Vector3 bias = new Vector3(0.5f, 0.1f, 0.3f);
Vector3 modified = original.WithRandomBiase(bias);
```

    -
    - **Parameters**:
        - `biasValue` (Vector3) - The maximum random bias for each component (x, y, z).
    - **Returns**: `Vector3` - A new `Vector3` with each component randomly biased by its corresponding value from `biasValue`.

Example:
```
// Adding a random bias to each component with different maximum
biases
Vector3 original = new Vector3(1, 2, 3);
Vector3 bias = new Vector3(0.5f, 0.1f, 0.3f);
Vector3 modified = original.WithRandomBiase(bias);
// modified might be (1.3, 2.1, 3.2), with different biases for each
component
```

2.

GetClosest

Description: Finds the closest position from a list of other positions to the current position. It calculates the squared distance to avoid the performance cost of taking square roots.

Usage:

```
Vector3 position = new Vector3(1, 2, 3);
IEnumerable<Vector3> otherPositions = new List<Vector3>
{
    new Vector3(4, 5, 6),
    new Vector3(7, 8, 9),
    new Vector3(1, 2, 4)
```

```
};
Vector3 closest = position.GetClosest(otherPositions);
```

- **Parameters**:
  - `otherPositions` (IEnumerable<Vector3>) - The collection of positions to compare against.
- **Returns**: `Vector3` - The closest position from the provided collection.

Example:

```
// Finding the closest position among a list of positions
Vector3 position = new Vector3(1, 2, 3);
var otherPositions = new List<Vector3>
{
```