Table of Contents

Namespace UniUtils.Data

Classes

Batcher
: Provides utility methods for processing items in batches with support for callbacks and error handling.

Crypter
: Provides methods for encrypting and decrypting strings using a simple XOR-based algorithm.

FadeTransitions
: Utility class for different transitions of various types.

FileHandle
: Represents a file handle that provides methods for file operations such as reading, writing, and deleting files.

FileHandleComparer
: Provides methods to compare two FileHandle objects for equality based on their full paths.

FileManager
: Provides utility methods for managing files and directories within different storage locations.

Formatters
: Provides utility methods for formatting values.

JsonObject<T>
: Represents a generic JSON object that can be serialized and deserialized using Unity's JsonUtility. The derived class is required to have the [Serializable] attribute.

LerpValue<T>
: A generic class for managing and interpolating a value towards a target value over time.

ObservableField<T>
: Represents an observable field that notifies subscribers when its value changes.

ObservableList<T>
: Represents a list that notifies subscribers when items are added or removed.

WaitHelper

Enums

EStorageLocation
: Specifies the storage location for files within the application.

Class Batcher

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Provides utility methods for processing items in batches with support for callbacks and error handling.

```
public static class Batcher
```

Inheritance

object ← Batcher

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ProcessInBatches<T>(IReadOnlyList<T>, Action<T>, int, float, Action<T>, Action<T, bool>, Action<IReadOnlyList<T>>, Action<IReadOnlyList<T>>, Action<IReadOnlyList<T>>, Action<T, Exception>)

Executes an action on a list of items in batches, with optional callbacks for various stages of processing.

```
public static IEnumerator ProcessInBatches<T>(IReadOnlyList<T> itemsList, Action<T>
actionOnItem, int itemsPerBatch = 5, float batchDelay = 0.1, Action<T> onItemProcessStart =
null, Action<T, bool> onItemProcessFinished = null, Action<IReadOnlyList<T>> onBatchStart =
null, Action<IReadOnlyList<T>> onBatchFinished = null, Action<IReadOnlyList<T>> onFinished =
null, Action<T, Exception> onError = null)
```

Parameters

`itemsList` IReadOnlyList<T>

The list of items to process.

`actionOnItem` Action<T>

The action to execute on each item.

`itemsPerBatch` int

The number of items to process per batch. Defaults to 5.

`batchDelay` float

The delay (in seconds) between processing batches. Defaults to 0.1f.

`onItemProcessStart` Action<T>

Optional callback invoked before processing each item; receives the item as `T`.

`onItemProcessFinished` Action<T, bool>

Optional callback invoked after processing each item; receives the item as `T` and a `bool` indicating success.

`onBatchStart` Action<IReadOnlyList<T>>

Optional callback invoked before processing each batch; receives the batch as `IReadOnlyList<T>`.

`onBatchFinished` Action<IReadOnlyList<T>>

Optional callback invoked when a batch is completed; receives the batch as `IReadOnlyList<T>`.

`onFinished` Action<IReadOnlyList<T>>

Optional callback invoked when all items have been processed; receives the full list of processed items as `IReadOnlyList<T>`.

`onError` Action<T, Exception>

Optional callback invoked when an error occurs during item processing; receives the item that failed (`T`) and the thrown `Exception`.

Returns

IEnumerator

An enumerator that can be used to execute the batches over time.

Type Parameters

`T`

The type of items in the list.

Examples

```
IReadOnlyList<int> numbers = new IReadOnlyList<int>() { 1, 2, 3, 4, 5, 6, 7 };

StartCoroutine(Batcher.ProcessInBatches(
    numbers,
    n => Debug.Log($"Processing: {n}"),
    itemsPerBatch: 3,
    batchDelay: 0.5f,
    onBatchStart: batch => Debug.Log($"Batch starting: {string.Join(", ", batch)}"),
    onBatchFinished: batch => Debug.Log($"Batch finished: {string.Join(", ", batch)}"),
    onFinished: all => Debug.Log("All items processed.")
));
```

Exceptions

System.ArgumentNullException

Thrown if `actionOnItem` is null.

ProcessPreBatched<T>(IReadOnlyList<IReadOnlyList<T>>, Action<T>, float, Action<T>, Action<T, bool>, Action<IReadOnlyList<T>>, Action<IReadOnlyList<T>>, Action<IReadOnlyList<T>>, Action<T, Exception>)

Executes an action on pre-batched lists of items, with optional callbacks for various stages of processing.

```
public static IEnumerator ProcessPreBatched<T>(IReadOnlyList<IReadOnlyList<T>>
preBatchedLists, Action<T> actionOnItem, float batchDelay = 0.1, Action<T>
onItemProcessStart = null, Action<T, bool> onItemProcessFinished = null,
Action<IReadOnlyList<T>> onBatchStart = null, Action<IReadOnlyList<T>> onBatchFinished =
null, Action<IReadOnlyList<T>> onFinished = null, Action<T, Exception> onError = null)
```

Parameters

`preBatchedLists` IReadOnlyList<IReadOnlyList<T>>

The pre-batched lists of items to process.

`actionOnItem` Action<T>

The action to execute on each item.

`batchDelay` float

The delay (in seconds) between processing batches. Defaults to 0.1f.

`onItemProcessStart` Action<T>

Optional callback invoked before processing each item; receives the item as `T`.

`onItemProcessFinished` Action<T, bool>

Optional callback invoked after processing each item; receives the item as `T` and a `bool` indicating success.

`onBatchStart` Action<IReadOnlyList<T>>

Optional callback invoked before processing each batch; receives the batch as `IReadOnlyList<T>`.

`onBatchFinished` Action<IReadOnlyList<T>>

Optional callback invoked when a batch is completed; receives the batch as `IReadOnlyList<T>`.

`onFinished` Action<IReadOnlyList<T>>

Optional callback invoked when all items have been processed; receives the full list of processed items as `IReadOnlyList<T>`.

`onError` Action<T, Exception>

Optional callback invoked when an error occurs during item processing; receives the item that failed (`T`) and the thrown `Exception`.

Returns

IEnumerator

An enumerator that can be used to execute the batches over time.

Type Parameters

`T`

The type of items in the batches.

Examples

```
IReadOnlyList<IReadOnlyList<int>> batches = new IReadOnlyList<IReadOnlyList<int>>()
{
    new IReadOnlyList<int>() { 1, 2, 3 },
    new IReadOnlyList<int>() { 4, 5, 6 },
    new IReadOnlyList<int>() { 7 }
};

StartCoroutine(Batcher.ProcessPreBatched(
    batches,
    n => Debug.Log($"Processing: {n}"),
    batchDelay: 0.5f,
    onBatchStart: batch => Debug.Log($"Batch starting: {string.Join(", ", batch)}"),
    onBatchFinished: batch => Debug.Log($"Batch finished: {string.Join(", ", batch)}"),
    onFinished: all => Debug.Log("All items processed.")
));
```

Exceptions

System.ArgumentNullException

Thrown if `actionOnItem` is null.

Class Crypter

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Provides methods for encrypting and decrypting strings using a simple XOR-based algorithm.

```
public class Crypter
```

Inheritance

object ← Crypter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Decrypt(byte[], string)

Decrypts a byte array using a XOR-based algorithm and returns the decrypted string.

```
public static string Decrypt(byte[] encryptedBytes, string encryptionKey)
```

Parameters

`encryptedBytes` byte[]

The byte array containing the encrypted data.

`encryptionKey` string

The encryption key used for the XOR operation.

Returns

string

The decrypted string.

Examples

```
string text = "HelloWorld";
string key = "k";
byte[] encrypted = Crypter.Encrypt(text, key);
string decrypted = Crypter.Decrypt(encrypted, key);
Debug.Log(decrypted); // "HelloWorld"
```

DecryptFromBase64(string, string)

Decrypts a Base64-encoded string using a XOR-based algorithm and returns the decrypted string.

```
public static string DecryptFromBase64(string encryptedBase64, string encryptionKey)
```

Parameters

`encryptedBase64` string

The Base64-encoded string containing the encrypted data.

`encryptionKey` string

The encryption key used for the XOR operation.

Returns

string

The decrypted string.

Examples

```
string key = "secret";
string encrypted = Crypter.EncryptToBase64("HiddenMessage", key);
string decrypted = Crypter.DecryptFromBase64(encrypted, key);
Debug.Log(decrypted); // Outputs: "HiddenMessage"
```

Encrypt(string, string)

Encrypts a string using a XOR-based algorithm and returns the encrypted data as a byte array.

```
public static byte[] Encrypt(string input, string encryptionKey)
```

Parameters

`input` string

The input string to encrypt.

`encryptionKey` string

The encryption key used for the XOR operation.

Returns

byte[]

A byte array containing the encrypted data.

Examples

```
string original = "Secret123";
string key = "key!";
byte[] encrypted = Crypter.Encrypt(original, key);
// encrypted contains binary data
```

EncryptToBase64(string, string)

Encrypts a string using a XOR-based algorithm and returns the encrypted data as a Base64-encoded string.

```
public static string EncryptToBase64(string input, string encryptionKey)
```

Parameters

`input` string

The input string to encrypt.

`encryptionKey` string

The encryption key used for the XOR operation.

Returns

string

A Base64-encoded string containing the encrypted data.

Examples

```
string secret = "MyPassword";
string key = "abc123";
string encryptedBase64 = Crypter.EncryptToBase64(secret, key);
Debug.Log(encryptedBase64); // Outputs something like: "GhoaFh1R..."
```

Enum EStorageLocation

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Specifies the storage location for files within the application.

```
public enum EStorageLocation
```

Fields

`DataPath = 1`

(Application.dataPath) Represents the application's data path, often used for accessing files bundled with the application.

`Persistent = 0`

(Application.persistentDataPath) Represents the persistent storage location, typically used for saving data that should remain across application sessions.

`StreamingAssets = 3`

(Application.streamingAssetsPath) Represents the streaming assets folder, often used for accessing read-only data bundled with the application.

`Temporary = 2`

(Application.temporaryCachePath) Represents a temporary storage location, typically used for storing data that does not need to persist.

Class FadeTransitions

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Utility class for different transitions of various types.

```
public class FadeTransitions : MonoBehaviour
```

Inheritance

object ← FadeTransitions

Methods

FadeLightColor(Light, Color, float)

Coroutine to fade a light's color over time.

```
public static IEnumerator FadeLightColor(Light targetLight, Color targetColor, float
duration = 1)
```

Parameters

`targetLight` Light

The light to modify.

`targetColor` Color

The target color to fade to.

`duration` float

The duration of the fade in seconds.

Returns

IEnumerator

An IEnumerator for use in a coroutine.

Examples

```
// Example: Fade a light's color to blue over 2 seconds.
StartCoroutine(FadeTransitions.FadeLightColor(
    targetLight: myLight,
    targetColor: Color.blue,
    duration: 2f
));
```

FadeLightIntensity(Light, float, float)

Coroutine to fade a light's intensity over time.

```
public static IEnumerator FadeLightIntensity(Light targetLight, float targetIntensity, float
duration = 1)
```

Parameters

`targetLight` Light

The light to modify.

`targetIntensity` float

The target intensity to fade to.

`duration` float

The duration of the fade in seconds.

Returns

IEnumerator

An IEnumerator for use in a coroutine.

Examples

```
// Example: Fade a light's intensity to 0 over 1.5 seconds.
StartCoroutine(FadeTransitions.FadeLightIntensity(
    targetLight: myLight,
    targetIntensity: 0f,
```

```
    duration: 1.5f
));
```

FadeMaterialColor(Material, Color, string, float)

Coroutine to fade a material's color property over time.

```
public static IEnumerator FadeMaterialColor(Material targetMaterial, Color targetColor,
string fieldName = "_EmissionColor", float duration = 1)
```

Parameters

`targetMaterial` Material

The material to modify.

`targetColor` Color

The target color to fade to.

`fieldName` string

The name of the color property to modify (default is "_EmissionColor").

`duration` float

The duration of the fade in seconds.

Returns

IEnumerator

An IEnumerator for use in a coroutine.

Examples

```
// Example: Fade a material's emission color to red over 3 seconds.
StartCoroutine(FadeTransitions.FadeMaterialColor(
    targetMaterial: myRenderer.material,
    targetColor: Color.red,
    fieldName: "_EmissionColor",
```

```
    duration: 3f
));
```

FadeValue<T>(Func<T>, Action<T>, T, Func<T, T, float, T>, float)

Generic coroutine to fade a value over time.

```
public static IEnumerator FadeValue<T>(Func<T> getter, Action<T> setter, T targetValue,
Func<T, T, float, T> lerpFunc, float duration = 1)
```

Parameters

`getter` Func<T>

Function to get the current value.

`setter` Action<T>

Action to set the new value.

`targetValue` T

The target value to fade to.

`lerpFunc` Func<T, T, float, T>

Function to interpolate between values.

`duration` float

The duration of the fade in seconds.

Returns

IEnumerator

An IEnumerator for use in a coroutine.

Type Parameters

`T`

The type of the value to fade.

Examples

```
// Example: Fade a float value from 0 to 1 over 2 seconds.
StartCoroutine(FadeTransitions.FadeValue(
    getter: () => someValue,
    setter: val => someValue = val,
    targetValue: 1f,
    lerpFunc: Mathf.Lerp,
    duration: 2f
));
```

Class FileHandle

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Represents a file handle that provides methods for file operations such as reading, writing, and deleting files.

```
public class FileHandle
```

Inheritance

object ← FileHandle

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

FileHandle(string, EStorageLocation)

Initializes a new instance of the FileHandle class with the specified relative path and storage location. Ensures the directory for the file exists.

```
public FileHandle(string relativePath, EStorageLocation location
= EStorageLocation.Persistent)
```

Parameters

`relativePath` string

The relative path of the file.

`location` EStorageLocation

The storage location. Defaults to Persistent.

Properties

Exists

```
public virtual bool Exists { get; }
```

Property Value

bool

Returns `true` if the file exists; otherwise, `false`.

Extension

```
public string Extension { get; }
```

Property Value

string

Returns the file extension.

FileName

```
public virtual string FileName { get; }
```

Property Value

string

The name of the file without the directory path.

FullPath

```
public virtual string FullPath { get; }
```

Property Value

string

The full path to the file, including the directory and file name.

IsReadOnly

```
public virtual bool IsReadOnly { get; }
```

Property Value

bool

Returns `true` if the file is read-only; otherwise, `false`.

Location

```
public virtual EStorageLocation Location { get; }
```

Property Value

EStorageLocation

The storage location of the file, such as Persistent.

RelativePath

```
public virtual string RelativePath { get; }
```

Property Value

string

The relative path of the file from the root of the specified storage location.

Methods

AppendText(string)

Appends the specified text content to the file.

```
public virtual bool AppendText(string content)
```

Parameters

`content` string

The text content to append to the file.

Returns

bool

`true` if the operation was successful; otherwise, `false`.

Examples

```
FileHandle handle = new FileHandle("log.txt");
handle.AppendText("Log entry\n");
```

Exceptions

System.IO.IOException

Thrown if appending to the file fails.

Copy(string, EStorageLocation?, bool)

Copies the current file to a new location and returns a new FileHandle instance for the copied file.

```
public virtual FileHandle Copy(string newRelativePath, EStorageLocation? location = null,
bool canOverwrite = false)
```

Parameters

`newRelativePath` string

The relative path for the new file location.

`location` EStorageLocation?

The storage location for the new file. If `null`, the current file's location is used.

`canOverwrite` bool

If `true`, allows overwriting an existing file at the new location; otherwise, throws an error if the target file already exists.

Returns

FileHandle

A new FileHandle instance representing the copied file, or `null` if the operation fails.

Examples

```
FileHandle file = new FileHandle("data/source.txt");
FileHandle copiedFile = file.Copy("data/destination.txt", canOverwrite: true);
```

Exceptions

System.ArgumentException

Thrown if the new relative path is invalid.

System.IO.FileNotFoundException

Thrown if the file does not exist.

System.IO.IOException

Thrown if writing to the file fails.

Delete()

Deletes the current file.

```
public virtual bool Delete()
```

Returns

bool

`true` if the file was successfully deleted; otherwise, `false`.

Examples

```
FileHandle handle = new FileHandle("temp/data.txt");
handle.Delete();
```

Exceptions

System.IO.IOException

Thrown if the delete operation fails.

Dump()

Logs the contents of the file to the Unity console.

```
public void Dump()
```

Examples

```
FileHandle handle = new FileHandle("logs/output.txt");
handle.Dump(); // Logs file contents to Unity console.
```

FileSizeBytes()

```
public virtual long FileSizeBytes()
```

Returns

long

Returns the size of the file in bytes, or 0 if the file does not exist.

LastModified()

```
public virtual DateTime? LastModified()
```

Returns

DateTime?

Returns the last modified date and time of the file, or `null` if the file does not exist.

Move(string, EStorageLocation?, bool)

Moves the current file to a new location and returns a new FileHandle instance for the moved file.

```
public virtual FileHandle Move(string newRelativePath, EStorageLocation? location = null,
bool canOverwrite = false)
```

Parameters

`newRelativePath` string

The relative path for the new file location.

`location` EStorageLocation?

`canOverwrite` bool

If `true`, allows overwriting an existing file at the new location; otherwise, throws an error if the target file already exists.

Returns

FileHandle

A new FileHandle instance representing the moved file, or `null` if the operation fails.

Examples

```
FileHandle file = new FileHandle("data/oldname.txt");
FileHandle movedFile = file.Move("data/newname.txt", canOverwrite: true);
```

Exceptions

System.IO.FileNotFoundException

Thrown if the file does not exist.

System.ArgumentException

Thrown if the new relative path is invalid.

System.IO.IOException

Thrown if the target file already exists.

System.IO.IOException

Thrown if writing to the file fails.

OpenReadStream()

Opens and returns a read-only FileStream for this file.

```
public virtual FileStream OpenReadStream()
```

Returns

FileStream

A System.IO.FileStream opened for read.

Examples

```
FileHandle handle = new FileHandle("data.txt");
using (FileStream stream = handle.OpenReadStream())
{
```

```
    // Read from stream
}
```

Exceptions

System.IO.FileNotFoundException

Thrown if the file does not exist.

OpenWriteStream(bool)

Opens and returns a write-only FileStream for this file.

```
public virtual FileStream OpenWriteStream(bool overwrite = true)
```

Parameters

`overwrite` bool

If true, the file will be recreated; if false, data will be appended.

Returns

FileStream

A System.IO.FileStream opened for write.

Examples

```
FileHandle handle = new FileHandle("log.txt");
using (FileStream stream = handle.OpenWriteStream(overwrite: true))
{
    byte[] bytes = Encoding.UTF8.GetBytes("Hello");
    stream.Write(bytes, 0, bytes.Length);
}
```

ReadBytes()

Reads the content of the file as a byte array.

```
public virtual byte[] ReadBytes()
```

Returns

byte[]

The content of the file as a byte array, or `null` if the file does not exist.

Examples

```
FileHandle handle = new FileHandle("data.bin");
byte[] data = handle.ReadBytes();
```

Exceptions

System.IO.IOException

Thrown if the file cannot be read.

ReadText()

Reads the content of the file as a string.

```
public virtual string ReadText()
```

Returns

string

The content of the file as a string, or `null` if the file does not exist.

Examples

```
FileHandle handle = new FileHandle("log.txt");
string content = handle.ReadText();
Debug.Log(content);
```

Exceptions

System.IO.IOException

Thrown if the file cannot be read.

Rename(string, bool)

Renames the current file by moving it to a new name within the same directory.

```
public virtual FileHandle Rename(string newName, bool canOverwrite = false)
```

Parameters

`newName` string

The new name for the file.

`canOverwrite` bool

If `true`, allows overwriting an existing file with the new name; otherwise, throws an error if the target file already exists.

Returns

FileHandle

A new FileHandle instance representing the renamed file, or `null` if the operation fails.

ToString()

Returns the full path of the file as a string.

```
public override string ToString()
```

Returns

string

The full path of the file.

WriteBytes(byte[])

Writes the specified byte array content to the file, overwriting any existing content.

```
public virtual bool WriteBytes(byte[] content)
```

Parameters

`content` byte[]

The byte array content to write to the file.

Returns

bool

`true` if the operation was successful; otherwise, `false`.

Examples

```
FileHandle handle = new FileHandle("binary.bin");
handle.WriteBytes(new byte[] { 1, 2, 3 });
```

Exceptions

System.IO.IOException

Thrown if writing to the file fails.

WriteText(string, Encoding)

Writes the specified text content to the file using the provided encoding, overwriting any existing content.

```
public virtual bool WriteText(string content, Encoding encoding)
```

Parameters

`content` string

The text content to write to the file.

`encoding` Encoding

The encoding to use when writing the text content.

Returns

bool

`true` if the operation was successful; otherwise, `false`.

Examples

```
FileHandle handle = new FileHandle("utf8.txt");
handle.WriteText("Hello", Encoding.UTF8);
```

Exceptions

System.IO.IOException

Thrown if writing to the file fails.

WriteText(string)

Writes the specified text content to the file, overwriting any existing content.

```
public virtual bool WriteText(string content)
```

Parameters

`content` string

The text content to write to the file.

Returns

bool

`true` if the operation was successful; otherwise, `false`.

Examples

```
FileHandle handle = new FileHandle("output.txt");
handle.WriteText("Hello, world!");
```

Exceptions

System.IO.IOException

Thrown if writing to the file fails.

Class FileHandleComparer

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Provides methods to compare two FileHandle objects for equality based on their full paths.

```
public class FileHandleComparer : IEqualityComparer<FileHandle>
```

Inheritance

object ← FileHandleComparer

Methods

Equals(FileHandle, FileHandle)

Determines whether the specified FileHandle objects are equal by comparing their full paths.

```
public bool Equals(FileHandle x, FileHandle y)
```

Parameters

`x` FileHandle

The first FileHandle to compare.

`y` FileHandle

The second FileHandle to compare.

Returns

bool

`true` if the full paths of both FileHandle objects are equal; otherwise, `false`.

GetHashCode(FileHandle)

```
public int GetHashCode(FileHandle obj)
```

Parameters

`obj` FileHandle

Returns

int

Class FileManager

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Provides utility methods for managing files and directories within different storage locations.

```
public static class FileManager
```

Inheritance

object ← FileManager

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ClearDirectory(string, EStorageLocation)

Deletes all contents of a directory without deleting the directory itself.

```
public static void ClearDirectory(string relativePath, EStorageLocation location
= EStorageLocation.Persistent)
```

Parameters

`relativePath` string

The relative path of the directory to clear.

`location` EStorageLocation

The storage location. Defaults to Persistent.

Examples

```
FileManager.ClearDirectory("Cache");
```

Exceptions

System.IO.IOException

Thrown if file or directory deletion fails.

System.UnauthorizedAccessException

Thrown if the operation lacks necessary permissions.

CopyDirectory(string, string, EStorageLocation)

Copies the contents of a source directory to a target directory within a specified storage location.

```
public static bool CopyDirectory(string sourceRelativePath, string targetRelativePath,
EStorageLocation location = EStorageLocation.Persistent)
```

Parameters

`sourceRelativePath` string

The relative path of the source directory.

`targetRelativePath` string

The relative path of the target directory.

`location` EStorageLocation

The storage location to use. Defaults to Persistent.

Returns

bool

`true` if the copy was successful; otherwise throws an exception.

Examples

```
FileManager.CopyDirectory("Configs", "Backup/Configs");
```

Exceptions

System.IO.DirectoryNotFoundException

Thrown if the source directory does not exist.

System.IO.IOException

Thrown if copying fails due to IO errors.

System.UnauthorizedAccessException

Thrown if the operation lacks necessary permissions.

DeleteDirectory(string, EStorageLocation)

Deletes a specified directory within a given storage location.

```
public static bool DeleteDirectory(string relativePath, EStorageLocation location
= EStorageLocation.Persistent)
```

Parameters

`relativePath` string

The relative path of the directory to delete.

`location` EStorageLocation

The storage location to use for resolving the directory path. Defaults to Persistent.

Returns

bool

`true` if the directory is deleted successfully; `false` if an error occurs.

Exceptions

System.IO.IOException

Thrown if file or directory deletion fails.

System.UnauthorizedAccessException

Thrown if the operation lacks necessary permissions.

EnsureDirectoryExists(string, EStorageLocation)

Ensures that a directory exists at the specified relative path within a given storage location. If the directory does not exist, it is created.

```
public static void EnsureDirectoryExists(string relativePath, EStorageLocation location
= EStorageLocation.Persistent)
```

Parameters

`relativePath` string

The relative path of the directory to check or create.

`location` EStorageLocation

The storage location to use for resolving the directory path. Defaults to Persistent.

GetFilesInDirectory(string, EStorageLocation)

Retrieves a list of files from a specified directory within a given storage location.

```
public static List<FileHandle> GetFilesInDirectory(string relativePath, EStorageLocation
location = EStorageLocation.Persistent)
```

Parameters

`relativePath` string

The relative path to the directory from which to retrieve files.

`location` EStorageLocation

The storage location to use for resolving the directory path. Defaults to Persistent.

Returns

List<FileHandle>

A list of FileHandle objects representing the files in the specified directory. If the directory does not exist, an empty list is returned.

Examples

```
List<FileHandle> files = FileManager.GetFilesInDirectory("Logs");
foreach (FileHandle file in files)
    file.Delete();
```

GetRootPath(EStorageLocation)

Retrieves the root path for the specified storage location.

```
public static string GetRootPath(EStorageLocation location)
```

Parameters

`location` EStorageLocation

The storage location for which to retrieve the root path.

Returns

string

The root path of the specified storage location.

GetSubdirectories(string, EStorageLocation)

Retrieves a list of subdirectories from a specified directory within a given storage location.

```
public static List<string> GetSubdirectories(string relativePath, EStorageLocation location
= EStorageLocation.Persistent)
```

Parameters

`relativePath` string

The relative path to the directory from which to retrieve subdirectories.

`location` EStorageLocation

The storage location to use for resolving the directory path. Defaults to Persistent.

Returns

List<string>

A list of relative paths representing the subdirectories in the specified directory. If the directory does not exist, an empty list is returned.

Examples

```
List<string> subdirs = FileManager.GetSubdirectories("Projects");
foreach (string dir in subdirs)
    Debug.Log(dir);
```

MoveDirectory(string, string, EStorageLocation)

Moves a directory from one path to another within a given storage location.

```
public static bool MoveDirectory(string fromRelativePath, string toRelativePath,
EStorageLocation location = EStorageLocation.Persistent)
```

Parameters

`fromRelativePath` string

The relative path of the source directory.

`toRelativePath` string

The relative path of the target directory.

`location` EStorageLocation

The storage location. Defaults to Persistent.

Returns

bool

`true` if the move succeeds; otherwise throws an exception.

Examples

```
FileManager.MoveDirectory("TempData", "Archived/TempData");
```

Exceptions

System.IO.IOException

Thrown if the target directory already exists or if the move fails.

TryIOAction(Action, string, Action<Exception, string>)

Executes an I/O action and handles any exceptions that occur during its execution.

```
public static bool TryIOAction(Action action, string errorContext, Action<Exception, string>
onError = null)
```

Parameters

`action` Action

The I/O action to execute.

`errorContext` string

A descriptive context for the error, used to provide additional information in the error message.

`onError` Action<Exception, string>

An optional callback to handle exceptions. The callback receives the exception and the formatted error message.

Returns

bool

`true` if the action executes successfully; `false` if an exception occurs.

Class Formatters

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Provides utility methods for formatting values.

```
public static class Formatters
```

Inheritance
object ← Formatters

Inherited Members
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

FormatDoubleToUsdString(double)

Formats a double value to a USD currency string.

```
public static string FormatDoubleToUsdString(double value)
```

Parameters

`value` double

The double value to format.

Returns

string

A formatted USD currency string.

Examples

```
double price = 49.99;
string formatted = Formatters.FormatDoubleToUsdString(price);
Debug.Log(formatted); // Output: "$49.99"
```

FormatFloatToTimeString(float)

Formats a float value representing time in seconds to a string in the format "MM:SS:FF".

```
public static string FormatFloatToTimeString(float time)
```

Parameters

`time` float

The time value in seconds.

Returns

string

A formatted time string in the format "MM:SS:FF".

Examples

```
float playTime = 125.37f; // 2 minutes, 5 seconds, 370 milliseconds
string formatted = Formatters.FormatFloatToTimeString(playTime);
Debug.Log(formatted); // Output: "02:05:37"
```

Class JsonObject<T>

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Represents a generic JSON object that can be serialized and deserialized using Unity's JsonUtility. The derived class is required to have the [Serializable] attribute.

```
public abstract class JsonObject<T>
```

Type Parameters

T

The type of the object to be deserialized.

Inheritance

object ← JsonObject<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Methods

FromJson(string)

```
public static T FromJson(string json)
```

Parameters

json string

Returns

T

Examples

```
// Define a serializable data model
[Serializable]
public class PlayerData : JsonObject<PlayerData>
{
    public string playerName;
    public int score;
}

// Create an instance and serialize it
PlayerData data = new PlayerData
{
    playerName = "Alex",
    score = 150
};

string json = data.ToJson();
Debug.Log(json);
// Output: {"playerName":"Alex","score":150}

// You can also deserialize it back:
PlayerData deserialized = PlayerData.FromJson(json);
Debug.Log(deserialized.playerName);
// Output: Alex
```

ToJson(bool)

Serializes the current object to a JSON string.

```
public string ToJson(bool prettyPrint = false)
```

Parameters

`prettyPrint` bool

If true, formats the JSON string with better readability.

Returns

string

A JSON string representation of the current object.

Examples

```
[Serializable]
public class PlayerData : JSONObject<PlayerData>
{
    public string playerName;
    public int score;
}

PlayerData data = new PlayerData { playerName = "Alex", score = 150 };
string json = data.ToJson();
Debug.Log(json); // Output: {"playerName":"Alex","score":150}
```

ToString()

Overrides the ToString method to return the JSON representation of the object.

```
public override string ToString()
```

Returns

string

A JSON string representation of the current object.

Class LerpValue<T>

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

A generic class for managing and interpolating a value towards a target value over time.

```
public sealed class LerpValue<T>
```

Type Parameters

T

The type of the value to interpolate.

Inheritance

object ← LerpValue<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

```
using UnityEngine;

public class ExampleUsage : MonoBehaviour
{
    private LerpValue<float> lerpAlpha;

    void Start()
    {
        // Start with 0, use Mathf.Lerp, and set speed to 2
        lerpAlpha = new LerpValue<float>(0f, Mathf.Lerp, 2f);
        lerpAlpha.SetTarget(1f); // Targeting full alpha
    }

    void Update()
    {
        lerpAlpha.Update(Time.deltaTime);
        float currentAlpha = lerpAlpha.Value;
        // Use currentAlpha for fading UI, etc.
```

```
    }
}
```

Constructors

LerpValue(T, Func<T, T, float, T>, float)

Initializes a new instance of the LerpValue<T> class.

```
public LerpValue(T initialValue, Func<T, T, float, T> lerpFunction, float speed = 5)
```

Parameters

`initialValue` T

The initial value of the interpolation.

`lerpFunction` Func<T, T, float, T>

The interpolation function to use.

`speed` float

The speed of interpolation (default is 5f).

Exceptions

System.ArgumentNullException

Thrown if `lerpFunction` is null.

Properties

LerpSpeed

Gets the speed at which the value interpolates towards the target.

```
public float LerpSpeed { get; }
```

Property Value

float

The speed of interpolation.

Target

Gets the target value.

```
public T Target { get; }
```

Property Value

T

The target value.

Value

Gets the current interpolated value.

```
public T Value { get; }
```

Property Value

T

The current value.

Methods

ForceSet(T)

Immediately sets the current and target values to a specified value.

```
public void ForceSet(T newValue)
```

Parameters

`newValue` T

The value to set.

SetLerpSpeed(float)

Sets the speed at which the value interpolates towards the target.

```
public void SetLerpSpeed(float newSpeed)
```

Parameters

`newSpeed` float

The new interpolation speed.

SetTarget(T)

Sets a new target value for interpolation.

```
public void SetTarget(T newTarget)
```

Parameters

`newTarget` T

The new target value.

Update(float)

Updates the current value by interpolating towards the target value.

```
public void Update(float deltaTime)
```

Parameters

`deltaTime` float

The time elapsed since the last update.

Class ObservableField<T>

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Represents an observable field that notifies subscribers when its value changes.

```
public class ObservableField<T>
```

Type Parameters

T

Inheritance

object ← ObservableField<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

The type of the value being observed.

```
ObservableField<int> observableInt = new ObservableField<int>(10);
observableInt.OnChange += newValue => Debug.Log($"Value changed to {newValue}");

observableInt.Value = 20; // Triggers OnChange and outputs: Value changed to 20
observableInt.Value = 20; // No event triggered because value is the same
```

Constructors

ObservableField(T)

Initializes a new instance of the ObservableField<T> class with an optional initial value.

```
public ObservableField(T initialValue = default)
```

Parameters

`initialValue` T

The initial value of the observable field.

Properties

Value

Gets or sets the value of the observable field.

```
public T Value { get; set; }
```

Property Value

T

Events

OnChange

Action to be invoked when the value changes.

```
public event Action<T> OnChange
```

Event Type

Action<T>

Class ObservableList<T>

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

Represents a list that notifies subscribers when items are added or removed.

```
public class ObservableList<T> : IEnumerable<T>
```

Type Parameters

T

The type of elements in the list.

Inheritance

object ← ObservableList<T>

Examples

```
ObservableList<string> observableList = new ObservableList<string>();
observableList.OnAdd += item => Debug.Log($"Added: {item}");
observableList.OnRemove += item => Debug.Log($"Removed: {item}");

observableList.Add("Hello");  // Output: Added: Hello
observableList.Add("World");  // Output: Added: World
observableList.Remove("Hello");  // Output: Removed: Hello
observableList.Clear();  // Output: Removed: World
```

Properties

Count

Gets the number of elements contained in the list.

```
public int Count { get; }
```

Property Value

int

this[int]

Gets the element at the specified index.

```
public T this[int index] { get; }
```

Parameters

`index` int

The zero-based index of the element to get.

Property Value

T

The element at the specified index.

Methods

Add(T)

Adds an item to the list and triggers the OnAdd event.

```
public void Add(T item)
```

Parameters

`item` T

The item to add to the list.

Clear()

Removes all items from the list and triggers the OnRemove event for each item.

```
public void Clear()
```

GetEnumerator()

Returns an enumerator that iterates through the list.

```
public IEnumerator<T> GetEnumerator()
```

Returns

IEnumerator<T>

An enumerator for the list.

Remove(T)

Removes the first occurrence of a specific item from the list and triggers the OnRemove event.

```
public bool Remove(T item)
```

Parameters

`item` T

The item to remove from the list.

Returns

bool

true if the item is successfully removed; otherwise, false.

Events

OnAdd

Event triggered when an item is added to the list.

```
public event Action<T> OnAdd
```

Event Type

Action<T>

OnRemove

Event triggered when an item is removed from the list.

```
public event Action<T> OnRemove
```

Event Type

Action<T>

Class WaitHelper

Namespace: UniUtils.Data

Assembly: cs.temp.dll.dll

```
public static class WaitHelper
```

Inheritance

object ← WaitHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

WaitForEndOfFrame

Gets a WaitForEndOfFrame instance.

```
public static WaitForEndOfFrame WaitForEndOfFrame { get; }
```

Property Value

WaitForEndOfFrame

Examples

```
// Usage example:
IEnumerator ExampleCoroutine()
{
    yield return WaitHelper.WaitForEndOfFrame;
}
```

WaitForFixedUpdate

Gets a WaitForFixedUpdate instance.

```
public static WaitForFixedUpdate WaitForFixedUpdate { get; }
```

Property Value

WaitForFixedUpdate

Examples

```
// Usage example:
IEnumerator ExampleCoroutine()
{
    yield return WaitHelper.WaitForFixedUpdate;
}
```

Methods

WaitForSeconds(float)

Gets a cached WaitForSeconds instance for the specified duration.

```
public static WaitForSeconds WaitForSeconds(float seconds)
```

Parameters

`seconds` float

The duration in seconds.

Returns

WaitForSeconds

A WaitForSeconds instance.

Examples

```
// Usage example:
IEnumerator ExampleCoroutine()
{
    // Wait for 1 second using cached instance
    yield return WaitHelper.WaitForSeconds(1f);
}
```

WaitForSecondsRealtime(float)

Gets a cached WaitForSecondsRealtime instance for the specified duration.

```
public static WaitForSecondsRealtime WaitForSecondsRealtime(float seconds)
```

Parameters

`seconds` float

The duration in seconds.

Returns

WaitForSecondsRealtime

A WaitForSecondsRealtime instance.

Examples

```
// Usage example:
IEnumerator ExampleCoroutine()
{
    // Wait for 1 second in real time using cached instance
    yield return WaitHelper.WaitForSecondsRealtime(1f);
}
```

Namespace UniUtils.Debugging

Classes

Logger
: A logger class that persists log entries and saves them to a file.

NavMeshPathDrawer
: Draws the path of a NavMeshAgent using a LineRenderer.

Structs

LogEntry
: Represents a log entry.

Enums

ELogTypeMask
: Represents a bitmask for categorizing log types in debugging.

Enum ELogTypeMask

Namespace: UniUtils.Debugging

Assembly: cs.temp.dll.dll

Represents a bitmask for categorizing log types in debugging.

```
public enum ELogTypeMask
```

Fields

`All = -1`

`Assert = 8`

`Error = 4`

`Exception = 16`

`Log = 1`

`None = 0`

`Warning = 2`

Struct LogEntry

Namespace: UniUtils.Debugging

Assembly: cs.temp.dll.dll

Represents a log entry.

```
public struct LogEntry
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

LogEntry(string, DateTime, string, string)

```
public LogEntry(string logType, DateTime time, string log, string stack)
```

Parameters

`logType` string

`time` DateTime

`log` string

`stack` string

Fields

dateTime

```
public DateTime dateTime
```

Field Value

logString

```
public string logString
```

Field Value

string

stackTrace

```
public string stackTrace
```

Field Value

string

type

```
public string type
```

Field Value

string

Class Logger

Namespace: UniUtils.Debugging

Assembly: cs.temp.dll.dll

A logger class that persists log entries and saves them to a file.

```
public class Logger : PersistentSingleton<Logger>
```

Inheritance

object ← Logger

Examples

```
Access the logger instance (singleton) and retrieve the current log as a string.
string currentLog = Logger.Instance.Log;

You can also customize ignored words and toggle stack trace appending in the inspector.
```

Properties

LogString

Gets the complete log as a string.

```
public string LogString { get; }
```

Property Value

string

Methods

Awake()

Initializes the logger.

```
protected override void Awake()
```

Events

OnNewLogEntry

```
public event Action<LogEntry> OnNewLogEntry
```

Event Type

Action<LogEntry>

Class NavMeshPathDrawer

Namespace: UniUtils.Debugging

Assembly: cs.temp.dll.dll

Draws the path of a NavMeshAgent using a LineRenderer.

```
public class NavMeshPathDrawer : MonoBehaviour
```

Inheritance

object ← NavMeshPathDrawer

Examples

```
Attach this component to a GameObject with a NavMeshAgent to visualize the agent's path.
Example usage:

1. Add NavMeshAgent component to your GameObject.
2. Add this NavMeshPathDrawer component.
3. Configure 'showPath', 'pathColor', and 'lineWidth' in the Inspector.

The path will be drawn automatically every frame.
```

Namespace UniUtils.Editor

Classes

ColliderBoundsDrawer

A MonoBehaviour that visualizes the bounds of a Collider in the Unity Editor.

Class ColliderBoundsDrawer

Namespace: UniUtils.Editor

Assembly: cs.temp.dll.dll

A MonoBehaviour that visualizes the bounds of a Collider in the Unity Editor.

```
public class ColliderBoundsDrawer : MonoBehaviour
```

Inheritance

object ← ColliderBoundsDrawer

Namespace UniUtils.EventSystem

Classes

EventChannel<TChannel>

Abstract class representing an event channel that can subscribe, unsubscribe, and publish events.

EventManager

Manages event channels and provides methods to subscribe and publish events.

Interfaces

IEvent<TChannel>

Interface for events with a specific event channel.

IEventChannel

Interface representing an event channel.

Class EventChannel<TChannel>

Namespace: UniUtils.EventSystem

Assembly: cs.temp.dll.dll

Abstract class representing an event channel that can subscribe, unsubscribe, and publish events.

```
public abstract class EventChannel<TChannel> : IEventChannel where TChannel
: EventChannel<TChannel>
```

Type Parameters

`TChannel`

The type of the event channel.

Inheritance

object ← EventChannel<TChannel>

Implements

IEventChannel

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Define a custom event channel

```
[Preserve] // Add Preserve to prevent the compiler from stripping the class.
public class PlayerEventChannel : EventChannel<PlayerEventChannel>
{
}
```

Fields

handlers

Dictionary to store event handlers by event type and handler ID.

```
protected readonly Dictionary<Type, Dictionary<int, Action<IEvent<TChannel>>>> handlers
```

Field Value

Dictionary<Type, Dictionary<int, Action<IEvent<TChannel>>>>

handlersToRemove

Set of handler IDs to be removed after publishing.

```
protected readonly HashSet<int> handlersToRemove
```

Field Value

HashSet<int>

isPublishing

Indicates whether an event is currently being published.

```
protected bool isPublishing
```

Field Value

bool

needsClearAfterPublishing

Indicates whether the handlers dictionary needs to be cleared after publishing.

```
protected bool needsClearAfterPublishing
```

Field Value

bool

nextHandlerId

The next handler ID to be assigned.

```
protected int nextHandlerId
```

Field Value

int

Methods

HandleHandlersToRemove(Dictionary<int, Action<IEvent<TChannel>>>)

Handles the removal of handlers that were marked for removal during publishing.

```
protected virtual void HandleHandlersToRemove(Dictionary<int, Action<IEvent<TChannel>>>
actions)
```

Parameters

`actions` Dictionary<int, Action<IEvent<TChannel>>>

The dictionary of actions to remove handlers from.

Publish<TEvent>(IEvent<TChannel>)

Publishes an event by invoking all handlers for the event type.

```
public virtual void Publish<TEvent>(IEvent<TChannel> @event) where TEvent : IEvent<TChannel>
```

Parameters

`event` IEvent<TChannel>

The event to publish.

Type Parameters

`TEvent`

The type of the event.

Subscribe<TEvent>(Action<TEvent>)

Subscribes a handler to an event type.

```
public virtual int Subscribe<TEvent>(Action<TEvent> handler) where TEvent : IEvent<TChannel>
```

Parameters

`handler` Action<TEvent>

The handler to be invoked when the event is published.

Returns

int

The ID of the subscribed handler.

Type Parameters

`TEvent`

The type of the event.

Unsubscribe<TEvent>(int)

Unsubscribes a handler from an event type.

```
public virtual void Unsubscribe<TEvent>(int handlerId) where TEvent : IEvent<TChannel>
```

Parameters

`handlerId` int

The ID of the handler to be unsubscribed.

Type Parameters

`TEvent`

The type of the event.

UnsubscribeAll()

Unsubscribes all event handlers.

```
public virtual void UnsubscribeAll()
```

Class EventManager

Namespace: UniUtils.EventSystem

Assembly: cs.temp.dll.dll

Manages event channels and provides methods to subscribe and publish events.

```
public class EventManager : PersistentSingleton<EventManager>
```

Inheritance

object ← EventManager

Examples

```
// Publisher example
public class Player : MonoBehaviour
{
    public void Jump()
    {
        PlayerJumpEvent jumpEvent = new PlayerJumpEvent
        {
            jumpStrength = 5.0f
        };
        EventManager.Publish<PlayerEventChannel, PlayerJumpEvent>(jumpEvent);
    }
}

// Subscriber example
public class SoundManager : MonoBehaviour
{
    private int handle;

    private void OnEnable()
    {
        handle = EventManager.Subscribe<PlayerEventChannel, PlayerJumpEvent>(OnPlayerJump);
    }

    private void OnDisable()
    {
        EventManager.Unsubscribe<PlayerEventChannel, PlayerJumpEvent>(handle);
    }
```

```
    private void OnPlayerJump(PlayerJumpEvent jumpEvent)
    {
        // React to jump event, e.g. play sound
    }
}
```

Fields

EventChannels

Dictionary to store event channels by their type.

```
protected static readonly Dictionary<Type, IEventChannel> EventChannels
```

Field Value

Dictionary<Type, IEventChannel>

Methods

Awake()

Called when the script instance is being loaded.

```
protected override void Awake()
```

GetEventChannel<T>()

Gets the event channel of the specified type.

```
protected static T GetEventChannel<T>() where T : EventChannel<T>
```

Returns

T

The event channel of the specified type.

Type Parameters

`T`

The type of the event channel.

Publish<TChannel, TEvent>(TEvent)

Publishes an event by invoking all handlers for the event type.

```
public static void Publish<TChannel, TEvent>(TEvent @event) where TChannel :
EventChannel<TChannel> where TEvent : IEvent<TChannel>
```

Parameters

`event` TEvent

The event to publish.

Type Parameters

`TChannel`

The type of the event channel.

`TEvent`

The type of the event.

RegisterEventChannels()

Registers all event channels by finding and instantiating classes that extend EventChannel.

```
protected static void RegisterEventChannels()
```

Subscribe<TChannel, TEvent>(Action<TEvent>)

Subscribes to an event by adding a handler to the event channel.

```
public static int Subscribe<TChannel, TEvent>(Action<TEvent> handler) where TChannel :
EventChannel<TChannel> where TEvent : IEvent<TChannel>
```

Parameters

`handler` Action<TEvent>

The handler to add for the event.

Returns

int

The ID of the subscribed handler.

Type Parameters

`TChannel`

The type of the event channel.

`TEvent`

The type of the event.

Unsubscribe<TChannel, TEvent>(int)

Unsubscribes from an event by removing a handler from the event channel.

```
public static void Unsubscribe<TChannel, TEvent>(int handlerId) where TChannel :
EventChannel<TChannel> where TEvent : IEvent<TChannel>
```

Parameters

`handlerId` int

The ID of the handler to remove for the event.

Type Parameters

`TChannel`

The type of the event channel.

`TEvent`

The type of the event.

UnsubscribeAll<TChannel>()

Unsubscribes all events by clearing the handlers dictionary in the specified event channel.

```
protected static void UnsubscribeAll<TChannel>() where TChannel : EventChannel<TChannel>
```

Type Parameters

`TChannel`

Interface IEvent<TChannel>

Namespace: UniUtils.EventSystem

Assembly: cs.temp.dll.dll

Interface for events with a specific event channel.

```
public interface IEvent<TChannel> where TChannel : EventChannel<TChannel>
```

Type Parameters

`TChannel`

The type of the event channel.

Examples

```
Define an event 'PlayerJumpEvent' on the 'PlayerEventChannel' channel

public class PlayerJumpEvent : IEvent<PlayerEventChannel>
{
    public float jumpStrength;
}
```

Interface IEventChannel

Namespace: UniUtils.EventSystem

Assembly: cs.temp.dll.dll

Interface representing an event channel.

```
public interface IEventChannel
```

Methods

UnsubscribeAll()

```
void UnsubscribeAll()
```

Namespace UniUtils.Extensions

Classes

AudioSourceExtension

Provides extension methods for Audio sources.

BoxColliderExtension

Provides extension methods for the BoxCollider class.

CameraExtension

Provides extension methods for the Camera.

ColorExtension

Provides extension methods for the Color class.

EnumExtension

Provides extensions for enums.

GameObjectExtension

Provides extension methods for the GameObject class.

LayerMaskExtension

Provides extension methods for the LayerMask class.

ListExtension

Provides extension methods for lists of floats and generic lists.

ObjectExtension

Provides extension methods for objects to enhance logging functionality.

QuaternionExtension

Provides extension methods for the Quaternion class.

RayExtension

Provides extensions for Rays.

RectExtension

RectTransformExtension

Provides extension methods for RectTransforms.

StringExtension

Provides extension methods for Strings.

TransformExtension

Provides extension methods for the Transform class.

Vector3Extension

Provides extension methods for the Vector3 class.

Class AudioSourceExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for Audio sources.

```
public static class AudioSourceExtension
```

Inheritance

object ← AudioSourceExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

FadeVolume(AudioSource, float, float, float?, Action)

Fades the volume of the given AudioSource from a start volume to a target volume over a specified duration.

```
public static IEnumerator FadeVolume(this AudioSource audioSource, float duration, float
targetVolume, float? startVolume = null, Action onFinished = null)
```

Parameters

`audioSource` AudioSource

The AudioSource to fade.

`duration` float

The duration over which to fade the volume.

`targetVolume` float

The target volume to reach at the end of the fade.

`startVolume` float?

The starting volume. Defaults to 0.

`onFinished` Action

An optional callback to invoke when the fade is complete.

Returns

IEnumerator

An IEnumerator that can be used to run the fade operation in a coroutine.

Examples

```
// Usage example:
IEnumerator FadeExample(AudioSource audioSource)
{
    // Fade volume to 0 over 2 seconds
    yield return audioSource.FadeVolume(2f, 0f);

    // Fade back to full volume over 1.5 seconds
    yield return audioSource.FadeVolume(1.5f, 1f);
}


// Usage example:
StartCoroutine(audioSource.FadeVolume(1.5f, 1f));
```

PlayAndWaitUntilFinished(AudioSource, AudioClip, float, Action)

Plays an AudioClip on the given AudioSource and waits until it finishes playing, with an optional offset to end playback early and a callback when finished.

```
public static IEnumerator PlayAndWaitUntilFinished(this AudioSource audioSource, AudioClip
clip, float endTimeOffset = 0, Action onFinished = null)
```

Parameters

`audioSource` AudioSource

The AudioSource to play the clip on.

`clip` AudioClip

The AudioClip to play.

`endTimeOffset` float

The time (in seconds) to offset from the end of the clip. Defaults to 0. If the offset is greater than or equal to the clip length, the method exits early.

`onFinished` Action

An optional callback to invoke when the clip finishes playing.

Returns

IEnumerator

An IEnumerator that can be used to run the operation in a coroutine.

Examples

```
// Usage example:
IEnumerator PlayClip(AudioSource audioSource, AudioClip clip)
{
    yield return audioSource.PlayAndWaitUntilFinished(clip, 0.2f, () =>
Debug.Log("Clip finished!"));
}
// StartCoroutine(PlayClip(audioSource, clip));
```

PlayOneShotWithVariance(AudioSource, AudioClip, float, float, float)

Plays an AudioClip with random pitch and volume variations for added audio diversity.

```
public static void PlayOneShotWithVariance(this AudioSource audioSource, AudioClip clip,
float audioVolume = 1, float maxPitchVariation = 0.1, float maxVolumeVariation = 0.1)
```

Parameters

`audioSource` AudioSource

The AudioSource to play the clip on.

`clip` AudioClip

The AudioClip to play.

`audioVolume` float

The base volume at which to play the clip. Defaults to 1.

`maxPitchVariation` float

The maximum variation in pitch. Defaults to 0.1.

`maxVolumeVariation` float

The maximum variation in volume. Defaults to 0.1.

Examples

```
// Usage example:
void PlayClipWithVariance(AudioSource audioSource, AudioClip clip)
{
    audioSource.PlayOneShotWithVariance(clip, 1f, 0.2f, 0.15f);
}
```

TransitionToClip(AudioSource, AudioClip, float?, float, Action)

Transitions the AudioSource to a new AudioClip by fading out the current clip (if playing), stopping it, and then fading in the new clip.

```
public static IEnumerator TransitionToClip(this AudioSource audioSource, AudioClip clip,
float? targetVolume = null, float transitionDuration = 1, Action onFinished = null)
```

Parameters

`audioSource` AudioSource

The AudioSource to transition.

`clip` AudioClip

The new AudioClip to play.

`targetVolume` float?

The target volume to fade to when playing the new clip. Defaults to the current volume of the AudioSource.

`transitionDuration` float

The total duration of the transition (fade out + fade in). Defaults to 1 second.

`onFinished` Action

An optional callback to invoke when the transition is finished.

Returns

IEnumerator

An IEnumerator that can be used to run the transition in a coroutine.

Examples

```
// Usage example:
IEnumerator TransitionExample(AudioSource audioSource, AudioClip newClip)
{
    yield return audioSource.TransitionToClip(newClip, 0.8f, 2f);
}
// StartCoroutine(TransitionExample(audioSource, newClip));
```

WaitUntilFinished(AudioSource, float, Action)

Waits until the current clip on the AudioSource finishes playing, with an optional offset to end early and a callback when finished.

```
public static IEnumerator WaitUntilFinished(this AudioSource audioSource, float
endTimeOffset = 0, Action onFinished = null)
```

Parameters

`audioSource` AudioSource

The AudioSource playing the clip.

`endTimeOffset` float

The time (in seconds) to offset from the end of the clip. Defaults to 0. If the offset is greater than or equal to the clip length, the method exits early.

`onFinished` Action

An optional callback to invoke when the clip finishes playing.

Returns

IEnumerator

An IEnumerator to be used in a coroutine.

Examples

```
yield return audioSource.WaitUntilFinished(0.1f, () => Debug.Log("Done"));
```

Class BoxColliderExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the BoxCollider class.

```
public static class BoxColliderExtension
```

Inheritance

object ← BoxColliderExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

RandomPoint(BoxCollider, bool)

Returns a random point within the bounds of the BoxCollider.

```
public static Vector3 RandomPoint(this BoxCollider collider, bool includeY = false)
```

Parameters

`collider` BoxCollider

`includeY` bool

If true, includes the Y coordinate in the random point; otherwise, uses the Y position of the collider's transform.

Returns

Vector3

A random point within the bounds of the BoxCollider.

Examples

```
// Usage example:
void Example(BoxCollider boxCollider)
{
    // Get a random point inside the collider ignoring Y (using collider's Y position)
    Vector3 point = boxCollider.RandomPoint();

    // Get a random point inside the collider including Y coordinate
    Vector3 pointWithY = boxCollider.RandomPoint(true);

    Debug.Log("Random point (no Y): " + point);
    Debug.Log("Random point (with Y): " + pointWithY);
}
```

Class CameraExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the Camera.

```
public static class CameraExtension
```

Inheritance

object ← CameraExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Capture(Camera, Action<byte[]>, LayerMask)

Captures the current camera view and invokes a callback with the image data as a byte array.

```
public static void Capture(this Camera camera, Action<byte[]> callback, LayerMask
excludedLayers = null)
```

Parameters

`camera` Camera

The Camera instance to capture the view from.

`callback` Action<byte[]>

The callback to invoke with the captured image data.

`excludedLayers` LayerMask

The layers to exclude from the camera capture. Defaults to none.

Examples

```
// Usage example:
void Start()
{
    Camera mainCam = Camera.main;
    mainCam.Capture(OnCaptured);
}

void OnCaptured(byte[] imageData)
{
    Debug.Log("Captured image byte size: " + imageData.Length);
    // You can now save the image or use it however you want
}
```

Raycast(Camera, out RaycastHit, LayerMask, Vector3?, float, QueryTriggerInteraction, bool)

Performs a raycast from the camera's screen point and returns whether a hit occurred.

```
public static bool Raycast(this Camera camera, out RaycastHit hit, LayerMask includedLayers,
Vector3? position = null, float maxDistance = 3.4028235E+38, QueryTriggerInteraction
queryTriggerInteraction = null, bool drawRay = false)
```

Parameters

`camera` Camera

The Camera instance to perform the raycast from.

`hit` RaycastHit

The RaycastHit object containing information about the hit.

`includedLayers` LayerMask

The layers to include in the raycast.

`position` Vector3?

The screen position to cast the ray from. Defaults to the center of the screen.

`maxDistance` float

The maximum distance for the raycast. Defaults to infinity.

`queryTriggerInteraction` QueryTriggerInteraction

Specifies whether to include trigger colliders in the raycast.

`drawRay` bool

Indicates whether to draw the ray in the scene view for debugging purposes.

Returns

bool

`true` if the raycast hit an object; otherwise, `false`.

Examples

Example usage:

```
RaycastHit hit;
bool didHit = Camera.main.Raycast(
    includedLayers: LayerMask.GetMask("Default"),
    hit: out hit,
    drawRay: true
);

if (didHit)
{
    Debug.Log($"Hit: {hit.collider.name}");
}
```

Class ColorExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the Color class.

```
public static class ColorExtension
```

Inheritance

object ← ColorExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

AdjustSaturation(Color, float)

Adjusts the saturation and brightness of the given color by the specified intensity.

```
public static Color AdjustSaturation(this Color color, float intensity)
```

Parameters

`color` Color

The original color to adjust.

`intensity` float

The intensity by which to adjust the saturation and value. Positive values increase, negative values decrease.

Returns

Color

A new color with adjusted saturation and value.

Examples

```
Color original = Color.red;
Color adjusted = original.AdjustSaturation(0.2f);
Debug.Log("Adjusted color: " + adjusted);
```

FromHex(string)

Converts a hexadecimal string to a Color.

```
public static Color FromHex(this string hex)
```

Parameters

`hex` string

The hexadecimal string to convert.

Returns

Color

The Color represented by the hexadecimal string.

Examples

```
string hex = "#FF00FF";
try
{
    Color color = hex.FromHex();
    Debug.Log("Color: " + color);
}
catch (Exception e)
{
    Debug.LogError(e.Message);
}
```

Exceptions

System.ArgumentException

System.ArgumentException: Thrown when the hex string is in an invalid format.

ToHex(Color)

Converts the color to a hexadecimal string.

```
public static string ToHex(this Color color)
```

Parameters

`color` Color

The color to convert.

Returns

string

A hexadecimal string representation of the color.

Examples

```
Color color = Color.green;
string hex = color.ToHex();
Debug.Log("Hex: " + hex);  // Output: #00FF00
```

WithAlpha(Color, float)

Creates a new color with the same RGB values but a modified alpha value.

```
public static Color WithAlpha(this Color color, float alpha)
```

Parameters

`color` Color

The original color.

`alpha` float

The new alpha value.

Returns

Color

A new color with the specified alpha value.

Examples

```
Color original = Color.blue;
Color withAlpha = original.WithAlpha(0.5f);
Debug.Log("Color with new alpha: " + withAlpha);
```

Class EnumExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extensions for enums.

```
public static class EnumExtension
```

Inheritance

object ← EnumExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

RandomValue<T>()

Returns a random value from the specified enum type.

```
public static T RandomValue<T>() where T : Enum
```

Returns

T

A random value from the enum.

Type Parameters

T

The enum type.

Examples

```
// Example enum
enum Colors { Red, Green, Blue }

// Get a random color
Colors randomColor = EnumExtension.RandomValue<Colors>();
Debug.Log(randomColor);
```

Class GameObjectExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the GameObject class.

```
public static class GameObjectExtension
```

Inheritance

object ← GameObjectExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetOrAddComponent<T>(GameObject)

Retrieves a component of the specified type from the GameObject. If the component does not exist, it adds a new one to the GameObject.

```
public static T GetOrAddComponent<T>(this GameObject obj) where T : Component
```

Parameters

`obj` GameObject

The GameObject to retrieve or add the component to.

Returns

T

The existing or newly added component of the specified type.

Type Parameters

`T`

The type of the component to retrieve or add.

Examples

```
// Usage example:
GameObject obj = new GameObject("MyObject");
Rigidbody rb = obj.GetOrAddComponent<Rigidbody>();
```

SetLayersRecursively(GameObject, LayerMask, bool)

Recursively sets the of this GameObject and all its children, returning a dictionary of each GameObject to its original layer mask.

```
public static Dictionary<GameObject, LayerMask> SetLayersRecursively(this GameObject obj,
LayerMask layerMask, bool includeInactive = true)
```

Parameters

`obj` GameObject

The root GameObject whose layer (and its descendants) will be set.

`layerMask` LayerMask

The target layer mask. Internally, is an int index, so this method uses `layerMask.value` as the new layer index.

`includeInactive` bool

If `true`, will include inactive children; otherwise only active ones.

Returns

Dictionary<GameObject, LayerMask>

A mapping each affected GameObject to its original (as a).

Class LayerMaskExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the LayerMask class.

```
public static class LayerMaskExtension
```

Inheritance

object ← LayerMaskExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

CreateLayerMaskFromIndices(params int[])

Creates a LayerMask from an array of layer indices.

```
public static LayerMask CreateLayerMaskFromIndices(params int[] layerIndices)
```

Parameters

`layerIndices` int[]

An array of layer indices.

Returns

LayerMask

A LayerMask representing the specified layers.

Examples

```
// Create a LayerMask for layers 0, 2, and 5
LayerMask mask = LayerMaskExtension.CreateLayerMaskFromIndices(0, 2, 5);
```

Exceptions

System.ArgumentException

Thrown when a layer index is out of the valid range (0-31).

CreateLayerMaskFromNames(params string[])

Creates a LayerMask from an array of layer names.

```
public static LayerMask CreateLayerMaskFromNames(params string[] layerNames)
```

Parameters

`layerNames` string[]

An array of layer names to convert into a LayerMask.

Returns

LayerMask

A LayerMask representing the specified layers.

Examples

```
// Create a LayerMask from layer names
LayerMask mask = LayerMaskExtension.NamesToLayer("Default", "UI", "Player");
```

GetSingleLayerIndex(LayerMask)

Retrieves the index of a single layer from the LayerMask.

```
public static int GetSingleLayerIndex(this LayerMask mask)
```

Parameters

`mask` LayerMask

The LayerMask to extract the layer index from.

Returns

int

The index of the single layer in the LayerMask. Returns 0 and logs an error if the LayerMask contains no layers or multiple layers.

Examples

```
LayerMask mask = LayerMaskExtension.CreateLayerMaskFromIndices(8);
int index = mask.GetSingleLayerIndex();  // index will be 8
```

Class ListExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for lists of floats and generic lists.

```
public static class ListExtension
```

Inheritance

object ← ListExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Center(List<Vector3>)

Calculates the center point of a list of Vector3s.

```
public static Vector3 Center(this List<Vector3> list)
```

Parameters

`list` List<Vector3>

The list of Vector3s to calculate the center point of.

Returns

Vector3

The center point of the list of Vector3s.

Examples

```
List<Vector3> positions = new()
{
    new Vector3(1, 0, 0),
    new Vector3(0, 1, 0),
    new Vector3(0, 0, 1)
};
Vector3 centerPoint = positions.Center();
// centerPoint is approximately (0.33, 0.33, 0.33)
```

Chunk<T>(List<T>, int)

Splits a list into smaller chunks of a specified size.

```
public static List<List<T>> Chunk<T>(this List<T> list, int chunkSize)
```

Parameters

`list` List<T>

The source list to split into chunks.

`chunkSize` int

The size of each chunk.

Returns

List<List<T>>

A list of smaller lists, each containing up to `chunkSize` items.

Type Parameters

`T`

The type of items in the list.

Clone<T>(List<T>)

Creates a shallow copy of the list.

```
public static List<T> Clone<T>(this List<T> list)
```

Parameters

`list` List<T>

The list to clone.

Returns

List<T>

A new list containing the same elements as the original list.

Type Parameters

`T`

The type of elements in the list.

Examples

```
List<float> original = new() { 1.0f, 2.0f, 3.0f };
List<float> copy = original.Clone();
// copy contains the same elements as original
```

Flatten<T>(List<List<T>>)

Flattens a list of lists into a single list by concatenating all inner lists.

```
public static List<T> Flatten<T>(this List<List<T>> listOfLists)
```

Parameters

`listOfLists` List<List<T>>

The list of lists to flatten.

Returns

List<T>

A single list containing all elements from the inner lists.

Type Parameters

`T`

The type of elements in the lists.

Examples

```
List<List<int>> nested = new()
{
    new List<int> { 1, 2 },
    new List<int> { 3, 4 },
    new List<int> { 5 }
};
List<int> flat = nested.Flatten();
// flat is { 1, 2, 3, 4, 5 }
```

Furthest(List<Transform>, Transform)

Finds the furthest transform in the list from the specified transform.

```
public static Transform Furthest(this List<Transform> transformList,
Transform compareTransform)
```

Parameters

`transformList` List<Transform>

The list of transforms to search.

`compareTransform` Transform

The transform to compare against.

Returns

Transform

The furthest transform from the specified transform.

Examples

```
List<Transform> enemies = new List<Transform>();
Transform player = someGameObject.transform;
Transform furthestEnemy = enemies.Furthest(player);
// furthestEnemy is the furthest enemy transform from the player
```

Lerp(List<float>, float)

Linearly interpolates between elements in a list of floats based on a parameter t.

```
public static float Lerp(this List<float> list, float t)
```

Parameters

`list` List<float>

The list of floats to interpolate.

`t` float

The interpolation parameter, typically between 0 and 1.

Returns

float

The interpolated float value.

Examples

```
List<float> points = new() { 0f, 10f, 20f };
float value = points.Lerp(0.25f);
// value is interpolated between 0 and 10, roughly 2.5
```

Exceptions

System.ArgumentException

System.ArgumentException: Thrown when the list is empty.

Nearest(List<Transform>, Transform)

Finds the nearest transform in the list to the specified transform.

```
public static Transform Nearest(this List<Transform> transformList,
Transform compareTransform)
```

Parameters

`transformList` List<Transform>

The list of transforms to search.

`compareTransform` Transform

The transform to compare against.

Returns

Transform

The nearest transform to the specified transform.

Examples

```
List<Transform> enemies = new List<Transform>();
Transform player = someGameObject.transform;
Transform nearestEnemy = enemies.Nearest(player);
// nearestEnemy is the closest enemy transform to the player
```

Next<T>(List<T>, T)

Returns the next element in the list after the specified item. If the item is not found, returns the first element if the list is not empty, otherwise returns the default value for the type.

```
public static T Next<T>(this List<T> list, T item)
```

Parameters

`list` List<T>

The list to search.

`item` T

The item to find the next element of.

Returns

T

The next element in the list after the specified item, or the first element if the item is not found and the list is not empty, otherwise the default value for the type.

Type Parameters

`T`

The type of elements in the list.

Examples

```
List<string> colors = new() { "Red", "Green", "Blue" };
string nextColor = colors.Next("Green");
// nextColor is "Blue"
string nextOfUnknown = colors.Next("Yellow");
// nextOfUnknown is "Red" (first element) because "Yellow" not found
```

ProcessInBatches<T>(List<T>, Action<T>, int, float, Action<T>, Action<T, bool>, Action<IReadOnlyList<T>>, Action<IReadOnlyList<T>>, Action<IReadOnlyList<T>>, Action<T, Exception>)

Executes a batch operation on a list of items, processing them in groups with optional delays and callbacks.

```
public static IEnumerator ProcessInBatches<T>(this List<T> itemsList, Action<T>
actionOnItem, int itemsPerBatch = 5, float batchDelay = 0.1, Action<T> onItemProcessStart =
```

```
  null, Action<T, bool> onItemProcessFinished = null, Action<IReadOnlyList<T>> onBatchStart =
  null, Action<IReadOnlyList<T>> onBatchFinished = null, Action<IReadOnlyList<T>> onFinished =
  null, Action<T, Exception> onError = null)
```

Parameters

`itemsList` List<T>

The list of items to process in batches.

`actionOnItem` Action<T>

The action to execute on each item in the batch.

`itemsPerBatch` int

The number of items to process in each batch. Default is 5.

`batchDelay` float

The delay (in seconds) between processing each batch. Default is 0.1f.

`onItemProcessStart` Action<T>

Optional callback invoked before processing each item; receives the item as `T`.

`onItemProcessFinished` Action<T, bool>

Optional callback invoked after processing each item; receives the item as `T` and a `bool` indicating success.

`onBatchStart` Action<IReadOnlyList<T>>

Optional callback invoked at the start of each batch; receives the current batch as a `IReadOnlyList<T>`.

`onBatchFinished` Action<IReadOnlyList<T>>

Optional callback invoked at the end of each batch; receives the processed batch as a `IReadOnlyList<T>`.

`onFinished` Action<IReadOnlyList<T>>

Optional callback invoked after all batches are processed; receives the entire list as a `IReadOnlyList<T>`.

`onError` Action<T, Exception>

Optional callback invoked when an error occurs during item processing; receives the item that failed (`T`) and the thrown `Exception`.

Returns

IEnumerator

An enumerator that can be used to execute the batch operation.

Type Parameters

`T`

The type of items in the list.

Examples

```
List<int> numbers = new() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
IEnumerator batch = numbers.ProcessInBatches(
    item => Debug.Log($"Processing: {item}"),
    itemsPerBatch: 3,
    batchDelay: 0.5f,
    onBatchStart: batchItems => Debug.Log($"Batch started: {string.Join(",
", batchItems)}"),
    onBatchFinished: batchItems => Debug.Log($"Batch finished: {string.Join(",
", batchItems)}"),
    onFinished: allItems => Debug.Log("All batches complete!")
);
// Use StartCoroutine(batch) in a MonoBehaviour to execute.
```

Random<T>(List<T>, int)

Returns a list of unique random elements from the list.

```
public static List<T> Random<T>(this List<T> list, int count)
```

Parameters

`list` List<T>

The list to select random elements from.

`count` int

The number of unique random elements to return.

Returns

List<T>

A list containing unique random elements from the original list.

Type Parameters

`T`

The type of elements in the list.

Examples

```
List<int> numbers = new() { 1, 2, 3, 4, 5 };
List<int> randomSubset = numbers.Random(3);
// randomSubset contains 3 unique elements randomly selected from numbers
```

Exceptions

System.ArgumentException

Thrown if count is greater than the list count or if count is negative.

Random<T>(List<T>)

Returns a random element from the list.

```
public static T Random<T>(this List<T> list)
```

Parameters

`list` List<T>

The list to select a random element from.

Returns

T

A random element from the list.

Type Parameters

`T`

The type of elements in the list.

Examples

```
List<int> values = new() { 10, 20, 30 };
int randomValue = values.Random();
// randomValue is one of 10, 20, or 30
```

Exceptions

System.ArgumentException

System.ArgumentException: Thrown when the list is empty.

Replace<T>(List<T>, T, T)

Replaces the first occurrence of a specified item in the list with a new item.

```
public static List<T> Replace<T>(this List<T> list, T oldItem, T newItem)
```

Parameters

`list` List<T>

The list in which the replacement will occur.

`oldItem` T

The item to be replaced.

`newItem` T

The item to replace the old item with.

Returns

List<T>

The original list with the specified item replaced.

Type Parameters

T

The type of elements in the list.

Examples

```
List<string> fruits = new() { "Apple", "Banana", "Cherry" };
fruits.Replace("Banana", "Orange");
// fruits is now { "Apple", "Orange", "Cherry" }
```

Shuffle<T>(List<T>)

Shuffles the elements of a list in place using the Fisher-Yates algorithm.

```
public static List<T> Shuffle<T>(this List<T> list)
```

Parameters

list List<T>

The list to shuffle.

Returns

List<T>

Type Parameters

T

The type of elements in the list.

Examples

```
List<string> names = new() { "Alice", "Bob", "Charlie" };
names.Shuffle();
// names order is now randomized, e.g., "Charlie", "Alice", "Bob"
```

SortInPlace<T>(List<T>, Comparison<T>)

Sorts the elements of the list in place using the specified comparison.

```
public static List<T> SortInPlace<T>(this List<T> list, Comparison<T> comparison)
```

Parameters

`list` List<T>

The list to sort.

`comparison` Comparison<T>

The comparison to use for sorting the elements.

Returns

List<T>

The sorted list.

Type Parameters

`T`

The type of elements in the list.

Examples

```
List<int> numbers = new() { 5, 3, 8, 1 };
numbers.SortInPlace((a, b) => a.CompareTo(b));
```

```
// numbers is now sorted: 1, 3, 5, 8
```

Class ObjectExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for objects to enhance logging functionality.

```
public static class ObjectExtension
```

Inheritance

object ← ObjectExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Log(object, object)

Logs a message with the object's type prefix.

```
public static void Log(this object obj, object message)
```

Parameters

`obj` object

The object to log the message for.

`message` object

The message to log.

Examples

```
this.Log("Hello, World!");
```

Log(object, params object[])

Logs multiple messages with the object's type prefix.

```
public static void Log(this object obj, params object[] messages)
```

Parameters

`obj` object

The object to log the messages for.

`messages` object[]

The messages to log.

Examples

```
this.Log("Health:", playerHealth, "Score:", playerScore);
```

LogError(object, object)

Logs an error message with the object's type prefix.

```
public static void LogError(this object obj, object message)
```

Parameters

`obj` object

The object to log the error message for.

`message` object

The error message to log.

Examples

```
this.LogError("Player died unexpectedly.");
```

LogError(object, params object[])

Logs multiple error messages with the object's type prefix.

```
public static void LogError(this object obj, params object[] messages)
```

Parameters

`obj` object

The object to log the error messages for.

`messages` object[]

The error messages to log.

Examples

```
this.LogError("Error code:", errorCode, "Details:", errorDetails);
```

LogException(object, Exception)

Logs an exception with the object's type prefix.

```
public static void LogException(this object obj, Exception exception)
```

Parameters

`obj` object

The object to log the exception for.

`exception` Exception

The exception to log.

Examples

```
try
{
```

```
    // some code that may throw
}
catch (Exception ex)
{
    this.LogException(ex);
}
```

LogException(object, string, Exception)

Logs an exception with a custom message and the object's type prefix.

```
public static void LogException(this object obj, string message, Exception exception)
```

Parameters

`obj` object

The object to log the exception for.

`message` string

The custom message to include with the exception.

`exception` Exception

The exception to log.

Examples

```
try
{
    // some code that may throw
}
catch (Exception ex)
{
    this.LogException("Custom error message", ex);
}
```

LogWarning(object, object)

Logs a warning message with the object's type prefix.

```
public static void LogWarning(this object obj, object message)
```

Parameters

`obj` object

The object to log the warning message for.

`message` object

The warning message to log.

Examples

```
this.LogWarning("Low health detected!");
```

LogWarning(object, params object[])

Logs multiple warning messages with the object's type prefix.

```
public static void LogWarning(this object obj, params object[] messages)
```

Parameters

`obj` object

The object to log the warning messages for.

`messages` object[]

The warning messages to log.

Examples

```
this.LogWarning("Low health:", health, "Ammo:", ammoCount);
```

Class QuaternionExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the Quaternion class.

```
public static class QuaternionExtension
```

Inheritance

object ← QuaternionExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

With(Quaternion, float?, float?, float?)

Returns a new Quaternion with the specified Euler angles applied.

```
public static Quaternion With(this Quaternion quaternion, float? x = null, float? y = null,
float? z = null)
```

Parameters

`quaternion` Quaternion

The original Quaternion.

`x` float?

The x component of the Euler angles to apply. Defaults to 0 if null.

`y` float?

The y component of the Euler angles to apply. Defaults to 0 if null.

`z` float?

The z component of the Euler angles to apply. Defaults to 0 if null.

Returns

Quaternion

A new Quaternion with the specified Euler angles applied.

Examples

```
Quaternion original = Quaternion.Euler(10, 20, 30);
Quaternion modified = original.With(y: 45);
Debug.Log(modified.eulerAngles); // Output: (10, 45, 30)
```

Class RayExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extensions for Rays.

```
public static class RayExtension
```

Inheritance

object ← RayExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Draw(Ray, Color, float, float)

Draws a ray in the Unity editor for debugging purposes.

```
public static void Draw(this Ray ray, Color color, float length = 100, float duration
= 0.25)
```

Parameters

`ray` Ray

The ray to draw.

`color` Color

The color of the ray.

`length` float

The length of the ray (default is 100f).

`duration` float

The duration the ray will be visible in seconds (default is 0.25f).

Examples

```
Ray ray = new Ray(transform.position, transform.forward);
ray.Draw(Color.red, 50f, 1f);
```

Class RectExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

```
public static class RectExtension
```

Inheritance

object ← RectExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

ToLocalSpace(Rect, Transform)

Converts a Rect from world space to local space.

```
public static Rect ToLocalSpace(this Rect r, Transform transform)
```

Parameters

`r` Rect

The Rect to convert.

`transform` Transform

The Transform to use for conversion.

Returns

Rect

A Rect in local space.

Examples

```
Rect worldRect = new Rect(0, 0, 1, 1);
Rect localRect = worldRect.ToLocalSpace(transform);
```

ToWorldSpace(Rect, Transform)

Converts a Rect from local space to world space.

```
public static Rect ToWorldSpace(this Rect r, Transform transform)
```

Parameters

`r` Rect

The Rect to convert.

`transform` Transform

The Transform to use for conversion.

Returns

Rect

A Rect in world space.

Examples

```
Rect localRect = new Rect(0, 0, 1, 1);
Rect worldRect = localRect.ToWorldSpace(transform);
```

Class RectTransformExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for RectTransforms.

```
public static class RectTransformExtension
```

Inheritance

object ← RectTransformExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

RebuildLayout(RectTransform)

Rebuilds the layout of the rect transform to fix potential sizing issues and UI glitches.

```
public static IEnumerator RebuildLayout(this RectTransform rect)
```

Parameters

`rect` RectTransform

The RectTransform of object.

Returns

IEnumerator

An IEnumerator for the coroutine.

Examples

```
StartCoroutine(myRectTransform.RebuildLayout());
```

Class StringExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for Strings.

```
public static class StringExtension
```

Inheritance

object ← StringExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Sanitize(string, string)

Sanitizes the input string by replacing spaces with underscores and removing special characters.

```
public static string Sanitize(this string text, string allowedPattern = "A-Za-z0-9_")
```

Parameters

`text` string

The input string to be sanitized.

`allowedPattern` string

A regex pattern defining the set of allowed characters. Defaults to "A-Za-z0-9_", which allows alphanumeric characters and underscores.

Returns

string

A sanitized string with spaces replaced by underscores and disallowed characters removed.

Examples

```
string sanitized = "Hello, World!".Sanitize();
// sanitized == "Hello_World"
```

Swap(string, string, string)

Swaps occurrences of two specified substrings within the given text.

```
public static string Swap(this string text, string swapA, string swapB)
```

Parameters

`text` string

The original text where the swap will occur.

`swapA` string

The first substring to swap.

`swapB` string

The second substring to swap.

Returns

string

A new string with the specified substrings swapped.

Examples

```
string result = "hello world".Swap("hello", "world");
// result == "world hello"
```

Truncate(string, int, string)

Truncates the input string to a specified maximum length without cutting off words. Appends a truncation indicator if necessary.

```
public static string Truncate(this string text, int maxLength, string indicator = "...")
```

Parameters

`text` string

The input string to be truncated.

`maxLength` int

The maximum length of the truncated string (not counting the indicator).

`indicator` string

The string to append if truncation occurs. Defaults to "...".

Returns

string

A truncated string that ends on a word boundary (space/tab/newline) with the truncation indicator appended. Returns the original string if it's shorter than or equal to maxLength. Returns an empty string if the input is null or maxLength is negative.

Examples

```
string t1 = "This is a long string".Truncate(10);
// t1 == "This is a…"

string t2 = "Short".Truncate(10);
// t2 == "Short"

string t3 = null.Truncate(5);
// t3 == ""

string t4 = "Supercalifragilistic".Truncate(5);
// t4 == "Super…"
```

Class TransformExtension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the Transform class.

```
public static class TransformExtension
```

Inheritance

object ← TransformExtension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

MoveAndRotateToTransform(Transform, Transform, float, bool)

Smoothly moves a Transform to match the position and rotation of a target Transform over a specified duration. Can optionally move in local space.

```
public static IEnumerator MoveAndRotateToTransform(this Transform target, Transform
destination, float duration = 0.5, bool useLocal = true)
```

Parameters

`target` Transform

The Transform to move.

`destination` Transform

The target Transform to move towards.

`duration` float

The duration of the movement in seconds. Defaults to 0.5 seconds.

`useLocal` bool

If true, operates on localPosition/localRotation instead of world position/rotation.

Returns

IEnumerator

An IEnumerator to be used in a coroutine.

Examples

```
StartCoroutine(characterTransform.MoveAndRotateToTransform(spawnTransform, 1.2f));
```

MoveToPosition(Transform, Vector3, float, bool)

Smoothly moves a Transform to a specified position over a given duration. Can optionally move in local space.

```
public static IEnumerator MoveToPosition(this Transform target, Vector3 endPosition, float
duration = 0.5, bool useLocal = true)
```

Parameters

`target` Transform

The Transform to move.

`endPosition` Vector3

The target position to move towards.

`duration` float

The duration of the movement in seconds. Defaults to 0.5 seconds.

`useLocal` bool

If true, operates on localPosition instead of world position.

Returns

IEnumerator

An IEnumerator to be used in a coroutine.

Examples

```
StartCoroutine(platformTransform.MoveToPosition(new Vector3(0f, 3f, 0f), 0.8f));
```

MoveToTransform(Transform, Transform, float, bool)

Smoothly moves a Transform to match the position of a target Transform over a specified duration. Can optionally move in local space.

```
public static IEnumerator MoveToTransform(this Transform target, Transform destination,
float duration = 0.5, bool useLocal = true)
```

Parameters

`target` Transform

The Transform to move.

`destination` Transform

The target Transform to move towards.

`duration` float

The duration of the movement in seconds. Defaults to 0.5 seconds.

`useLocal` bool

If true, operates on localPosition instead of world position.

Returns

IEnumerator

An IEnumerator to be used in a coroutine.

Examples

```
StartCoroutine(enemyTransform.MoveToTransform(playerTransform, 1f, false));
StartCoroutine(handTransform.MoveToTransform(itemSlotTransform, 0.5f));
```

Raycast(Transform, Vector3, out RaycastHit, LayerMask, Vector3, Vector3, float, QueryTriggerInteraction, bool)

Performs a raycast from the Transform's position towards a target position, with optional offsets and settings.

```
public static bool Raycast(this Transform transform, Vector3 targetPosition, out RaycastHit hit, LayerMask includedLayers, Vector3 localRayOffset = null, Vector3 targetRayOffset = null, float maxDistance = 3.4028235E+38, QueryTriggerInteraction queryTriggerInteraction = null, bool drawRay = false)
```

Parameters

`transform` Transform

The Transform from which the raycast originates.

`targetPosition` Vector3

The position to raycast towards.

`hit` RaycastHit

The RaycastHit object that will store information about the hit.

`includedLayers` LayerMask

The layers to include in the raycast.

`localRayOffset` Vector3

An optional offset applied to the ray's origin.

`targetRayOffset` Vector3

An optional offset applied to the ray's target position.

`maxDistance` float

The maximum distance for the raycast. Defaults to float.MaxValue.

`queryTriggerInteraction` QueryTriggerInteraction

Specifies whether the raycast should interact with trigger colliders.

`drawRay` bool

Whether to draw the ray in the scene for debugging purposes.

Returns

bool

True if the raycast hits an object; otherwise, false.

Examples

```
RaycastHit hit;
if(transform.Raycast(target.position, out hit, LayerMask.GetMask("Default")))
{
    Debug.Log("Hit object: " + hit.collider.name);
}
```

RaycastDirection(Transform, Vector3, out RaycastHit, LayerMask, Vector3, float, QueryTriggerInteraction, bool)

Performs a raycast from the Transform's position in a specified direction, with optional offsets and settings.

```
public static bool RaycastDirection(this Transform transform, Vector3 direction, out
RaycastHit hit, LayerMask includedLayers, Vector3 localRayOffset = null, float maxDistance =
3.4028235E+38, QueryTriggerInteraction queryTriggerInteraction = null, bool drawRay = false)
```

Parameters

`transform` Transform

The Transform from which the raycast originates.

`direction` Vector3

The direction in which to raycast.

`hit` RaycastHit

The RaycastHit object that will store information about the hit.

`includedLayers` LayerMask

The layers to include in the raycast.

`localRayOffset` Vector3

An optional offset applied to the ray's origin.

`maxDistance` float

The maximum distance for the raycast. Defaults to float.MaxValue.

`queryTriggerInteraction` QueryTriggerInteraction

Specifies whether the raycast should interact with trigger colliders.

`drawRay` bool

Whether to draw the ray in the scene for debugging purposes.

Returns

bool

True if the raycast hits an object; otherwise, false.

Examples

```
RaycastHit hit;
Vector3 direction = transform.up;
if(transform.RaycastDirection(direction, out hit, LayerMask.GetMask("Default"),
maxDistance: 10f))
{
    Debug.Log("Hit object in up direction: " + hit.collider.name);
}
```

RotateToRotation(Transform, Quaternion, float, bool)

Smoothly rotates a Transform to a specified Quaternion rotation over a given duration. Can optionally rotate in local space.

```
public static IEnumerator RotateToRotation(this Transform target, Quaternion endRot, float
duration = 0.5, bool useLocal = true)
```

Parameters

`target` Transform

The Transform to rotate.

`endRot` Quaternion

The target rotation as a Quaternion.

`duration` float

The duration of the rotation in seconds. Defaults to 0.5 seconds.

`useLocal` bool

If true, operates on localRotation instead of world rotation.

Returns

IEnumerator

An IEnumerator to be used in a coroutine.

Examples

```
StartCoroutine(propellerTransform.RotateToRotation(Quaternion.Euler(0f, 0f, 360f),
2f, false));
StartCoroutine(cameraTransform.RotateToRotation(Quaternion.Euler(15f, 0f, 0f), 1f, true));
```

RotateToTransform(Transform, Transform, float, bool)

Smoothly rotates a Transform to match another Transform's rotation over a specified duration. Can optionally rotate in local space.

```
public static IEnumerator RotateToTransform(this Transform target, Transform destination,
float duration = 0.5, bool useLocal = true)
```

Parameters

`target` Transform

The Transform to rotate.

`destination` Transform

The Transform whose rotation to match.

`duration` float

The duration of the rotation in seconds. Defaults to 0.5 seconds.

`useLocal` bool

If true, operates on localRotation instead of world rotation.

Returns

IEnumerator

An IEnumerator to be used in a coroutine.

Examples

```
StartCoroutine(turretTransform.RotateToTransform(enemyTransform, 0.7f, false));
```

Class Vector3Extension

Namespace: UniUtils.Extensions

Assembly: cs.temp.dll.dll

Provides extension methods for the Vector3 class.

```
public static class Vector3Extension
```

Inheritance

object ← Vector3Extension

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

DistanceTo(Vector3, Vector3)

Calculates the distance between two vectors.

```
public static float DistanceTo(this Vector3 vector, Vector3 other)
```

Parameters

`vector` Vector3

The original vector.

`other` Vector3

The vector to calculate the distance to.

Returns

float

The distance between the two vectors.

Examples

```
Vector3 a = new Vector3(0, 0, 0);
Vector3 b = new Vector3(3, 4, 0);
float dist = a.DistanceTo(b);  // Result: 5
```

Divide(Vector3, float)

Divides each component of the vector by a scalar value.

```
public static Vector3 Divide(this Vector3 vector, float value)
```

Parameters

`vector` Vector3

The original vector.

`value` float

The scalar value to divide by.

Returns

Vector3

A new vector with each component divided by the scalar value, or the original component if the divisor is zero.

Examples

```
Vector3 v = new Vector3(10, 20, 30);
Vector3 result = v.Divide(2);  // Result: (5, 10, 15)
```

Divide(Vector3, Vector3)

Divides each component of the vector by the corresponding component of another vector.

```
public static Vector3 Divide(this Vector3 vector, Vector3 other)
```

Parameters

`vector` Vector3

The original vector.

`other` Vector3

The vector to divide by.

Returns

Vector3

A new vector with each component divided by the corresponding component of the other vector, or the original component if the divisor is zero.

Examples

```
Vector3 a = new Vector3(10, 20, 30);
Vector3 b = new Vector3(2, 0, 5);
Vector3 result = a.Divide(b);  // Result: (5, 20, 6) – y unchanged because divisor is 0
```

Multiply(Vector3, float)

Multiplies each component of the vector by a scalar value.

```
public static Vector3 Multiply(this Vector3 vector, float value)
```

Parameters

`vector` Vector3

The original vector.

`value` float

The scalar value to multiply with.

Returns

Vector3

A new vector with each component multiplied by the scalar value.

Examples

```
Vector3 v = new Vector3(1, 2, 3);
Vector3 result = v.Multiply(2);  // Result: (2, 4, 6)
```

Multiply(Vector3, Vector3)

Multiplies each component of the vector by the corresponding component of another vector.

```
public static Vector3 Multiply(this Vector3 vector, Vector3 other)
```

Parameters

`vector` Vector3

The original vector.

`other` Vector3

The vector to multiply with.

Returns

Vector3

A new vector with each component multiplied by the corresponding component of the other vector.

Examples

```
Vector3 a = new Vector3(2, 3, 4);
Vector3 b = new Vector3(5, 6, 7);
Vector3 result = a.Multiply(b);  // Result: (10, 18, 28)
```

With(Vector3, float?, float?, float?)

Creates a new vector with the specified components replaced by the provided values.

```
public static Vector3 With(this Vector3 vector, float? x = null, float? y = null, float? z
= null)
```

Parameters

`vector` Vector3

The original vector.

`x` float?

The new x-component value, or null to keep the original x-component.

`y` float?

The new y-component value, or null to keep the original y-component.

`z` float?

The new z-component value, or null to keep the original z-component.

Returns

Vector3

A new vector with the specified components replaced by the provided values.

Examples

```
Vector3 v = new Vector3(1, 2, 3);
Vector3 modified = v.With(y: 10);  // Result: (1, 10, 3)
```

Namespace UniUtils.FSM

Classes

FunctionPredicate

A concrete implementation of IPredicate that uses a function delegate to evaluate a condition.

NotPredicate

Represents a predicate that negates the result of another predicate.

StateMachine

Represents a state machine that manages state transitions and updates.

StateNode

Represents a node in the state machine, holding a state and its transitions.

Transition

Implements a transition between states in a state machine.

Interfaces

IPredicate

Represents a predicate that can be evaluated to determine a boolean condition.

IState

Represents a state in the state machine.

ITransition

Represents a transition between states in a state machine.

Class FunctionPredicate

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

A concrete implementation of IPredicate that uses a function delegate to evaluate a condition.

```
public class FunctionPredicate : IPredicate
```

Inheritance

object ← FunctionPredicate

Implements

IPredicate

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Example usage:

```
bool isHealthLow = false;
IPredicate lowHealthPredicate = new FunctionPredicate(() => isHealthLow);

if (lowHealthPredicate.Evaluate())
{
    Debug.Log("Health is low!");
}
```

Constructors

FunctionPredicate(Func<bool>)

```
public FunctionPredicate(Func<bool> predicate)
```

Parameters

`predicate` Func<bool>

Methods

Evaluate()

```
public bool Evaluate()
```

Returns

bool

Operators

operator !(FunctionPredicate)

```
public static IPredicate operator !(FunctionPredicate original)
```

Parameters

`original` FunctionPredicate

Returns

IPredicate

Interface IPredicate

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

Represents a predicate that can be evaluated to determine a boolean condition.

```
public interface IPredicate
```

Methods

Evaluate()

```
bool Evaluate()
```

Returns

bool

Interface IState

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

Represents a state in the state machine.

```
public interface IState
```

Examples

Example implementation of a state:

```
public class IdleState : IState
{
    public void Enter()
    {
        Debug.Log("Entering Idle State");
    }

    public void Update()
    {
        // Idle behavior
    }

    public void FixedUpdate()
    {
        // Physics-related idle behavior
    }

    public void Exit()
    {
        Debug.Log("Exiting Idle State");
    }
}
```

Methods

Enter()

Called when the state is entered.

```
void Enter()
```

Exit()

Called when the state is exited.

```
void Exit()
```

FixedUpdate()

Called at fixed intervals to update the state.

```
void FixedUpdate()
```

Update()

Called every frame to update the state.

```
void Update()
```

Interface ITransition

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

Represents a transition between states in a state machine.

```
public interface ITransition
```

Properties

Predicate

Gets the predicate that determines if the transition should occur.

```
IPredicate Predicate { get; }
```

Property Value

IPredicate

TargetState

Gets the target state of the transition.

```
IState TargetState { get; }
```

Property Value

IState

Class NotPredicate

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

Represents a predicate that negates the result of another predicate.

```
public class NotPredicate : IPredicate
```

Inheritance

object ← NotPredicate

Implements

IPredicate

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

NotPredicate(IPredicate)

```
public NotPredicate(IPredicate original)
```

Parameters

`original` IPredicate

Methods

Evaluate()

Evaluates the predicate by negating the result of the original predicate.

```
public bool Evaluate()
```

Returns

bool

`true` if the original predicate evaluates to `false`; otherwise, `false`.

Class StateMachine

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

Represents a state machine that manages state transitions and updates.

```
public class StateMachine
```

Inheritance

object ← StateMachine

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Example usage:

```
public class IdleState : IState
{
    public void Enter() => Debug.Log("Entered Idle");
    public void Update() => Debug.Log("Updating Idle");
    public void FixedUpdate() { }
    public void Exit() => Debug.Log("Exited Idle");
}

public class MoveState : IState
{
    public void Enter() => Debug.Log("Entered Move");
    public void Update() => Debug.Log("Updating Move");
    public void FixedUpdate() { }
    public void Exit() => Debug.Log("Exited Move");
}

void SetupStateMachine()
{
    IdleState idle = new IdleState();
    MoveState move = new MoveState();
    StateMachine stateMachine = new StateMachine();
```

```
    // Add transitions
    stateMachine.AddTransition(idle, move, new FunctionPredicate(()
=> Input.GetKey(KeyCode.Space)));
    stateMachine.AddTransition(move, idle, new FunctionPredicate(() =>
!Input.GetKey(KeyCode.Space)));

    // Start in Idle
    stateMachine.SetState(idle);

    // In your MonoBehaviour.Update():
    stateMachine.Update();
}
```

Properties

CurrentState

```
public StateNode CurrentState { get; }
```

Property Value

StateNode

Methods

AddAnyTransition(IState, IPredicate)

Adds a transition from any state to a specified state with a specified predicate.

```
public void AddAnyTransition(IState to, IPredicate predicate)
```

Parameters

`to` IState

The state to transition to.

`predicate` IPredicate

The predicate that determines if the transition should occur.

AddTransition(IState, IState, IPredicate)

Adds a transition from one state to another with a specified predicate.

```
public void AddTransition(IState from, IState to, IPredicate predicate)
```

Parameters

`from` IState

The state to transition from.

`to` IState

The state to transition to.

`predicate` IPredicate

The predicate that determines if the transition should occur.

FixedUpdate()

Calls the FixedUpdate method on the current state.

```
public void FixedUpdate()
```

SetState(IState)

Sets the current state of the state machine.

```
public void SetState(IState state)
```

Parameters

`state` IState

The new state to set.

Update()

Updates the state machine, checking for transitions and updating the current state.

```
public void Update()
```

Events

OnStateChanged

```
public event Action<IState> OnStateChanged
```

Event Type

Action<IState>

Class StateNode

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

Represents a node in the state machine, holding a state and its transitions.

```
public sealed class StateNode
```

Inheritance

object ← StateNode

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

StateNode(IState)

Initializes a new instance of the StateNode class with the specified state.

```
public StateNode(IState state)
```

Parameters

`state` IState

The state to associate with this node.

Properties

State

Gets the state associated with this node.

```
public IState State { get; }
```

Property Value

IState

Transitions

Gets the set of transitions from this state.

```
public HashSet<ITransition> Transitions { get; }
```

Property Value

HashSet<ITransition>

Methods

AddTransition(IState, IPredicate)

Adds a transition from this state to another state with a specified predicate.

```
public void AddTransition(IState to, IPredicate predicate)
```

Parameters

`to` IState

The state to transition to.

`predicate` IPredicate

The predicate that determines if the transition should occur.

Class Transition

Namespace: UniUtils.FSM

Assembly: cs.temp.dll.dll

Implements a transition between states in a state machine.

```
public class Transition : ITransition
```

Inheritance

object ← Transition

Implements

ITransition

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Examples

Example usage:

```
void CreateTransition()
{
    IState idle = new IdleState();
    IState move = new MoveState();

    IPredicate canMove = new FunctionPredicate(() => Input.GetKey(KeyCode.Space));

    ITransition transition = new Transition(canMove, move);

    // Example: use in a state machine
    StateMachine stateMachine = new StateMachine();
    stateMachine.AddTransition(idle, move, canMove);
}
```

Constructors

Transition(IPredicate, IState)

Initializes a new instance of the Transition class.

```
public Transition(IPredicate predicate, IState targetState)
```

Parameters

`predicate` IPredicate

The predicate that determines if the transition should occur.

`targetState` IState

The target state of the transition.

Properties

Predicate

Gets the predicate that determines if the transition should occur.

```
public IPredicate Predicate { get; }
```

Property Value

IPredicate

TargetState

Gets the target state of the transition.

```
public IState TargetState { get; }
```

Property Value

IState

Namespace UniUtils.GameObjects

Classes

CacheDictionary<TKey, T>

Represents a dictionary that caches items and provides methods to manage the cache.

ColliderEventHandler

Handles runtime registration of collision and trigger events based on object tags.

EphemeralSingleton<T>

An ephemeral singleton that does not persist across scene loads.

GenericObjectPool<T>

A generic object pool for managing reusable objects of type T.

PersistentObject

A Unity component that assigns a unique GUID to the object for persistent identification.

PersistentSingleton<T>

A persistent singleton that is not destroyed on scene load.

Singleton<T>

Abstract base class for creating singleton MonoBehaviour instances.

TransformAnimator

A component that animates the position, rotation, and scale of a target Transform based on configurable animation settings.

TransformAnimator.TransformAnimatorComponentData

Represents the configuration data for animating a specific transform component.

Structs

ColliderEvent

Struct representing a collider event with callbacks for enter and exit interactions.

ColliderEventEntry

Stores collision and trigger events.

Interfaces

IPoolable

Interface for objects that can be managed by a pool. Provides methods to handle pooling, recycling, and returning operations.

Class CacheDictionary<TKey, T>

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

Represents a dictionary that caches items and provides methods to manage the cache.

```
public abstract class CacheDictionary<TKey, T> : EphemeralSingleton<CacheDictionary<TKey,
T>>
```

Type Parameters

`TKey`

The type of the key used in the dictionary.

`T`

The type of the value stored in the dictionary.

Inheritance

object ← Singleton<CacheDictionary<TKey, T>> ← EphemeralSingleton<CacheDictionary<TKey, T>> ← CacheDictionary<TKey, T>

Inherited Members

Singleton<CacheDictionary<TKey, T>>.instance , Singleton<CacheDictionary<TKey, T>>.Instance , Singleton<CacheDictionary<TKey, T>>.Awake()

Examples

Example of using a cache to store surface types based on a ground transform:

```
public class SurfaceType
{
    public string type;
}

public class GroundSurfaceCache : CacheDictionary<Transform, string> { }

public class Player
{
    public void OnFootstep(Transform groundTransform)
```

```
    {
        string surfaceType = GroundSurfaceCache.Instance.GetOrAdd(
            groundTransform,
            transform => transform.GetComponent<SurfaceType>()?.type,
            discardNullValue: true
        );
        Debug.Log("Footstep on surface: " + surfaceType);
    }
}
```

Methods

AddCachedItem(TKey, T)

Adds or updates an item in the cache.

```
public void AddCachedItem(TKey key, T value)
```

Parameters

`key` TKey

The key of the item to add or update.

`value` T

The value to associate with the key.

ClearCache()

Clears all items from the cache.

```
public void ClearCache()
```

ContainsKey(TKey)

Determines whether the cache contains the specified key.

```
public bool ContainsKey(TKey key)
```

Parameters

`key` TKey

The key to check for existence.

Returns

bool

`true` if the key exists in the cache; otherwise, `false`.

GetOrAdd(TKey, Func<TKey, T>, bool)

Retrieves a value from the cache or adds it using the specified factory function if it does not exist.

```
public T GetOrAdd(TKey key, Func<TKey, T> valueFactory, bool discardNullValue = true)
```

Parameters

`key` TKey

The key to retrieve or add.

`valueFactory` Func<TKey, T>

The function used to create the value if it does not exist.

`discardNullValue` bool

Indicates whether to discard null values. Defaults to `true`.

Returns

T

The value associated with the key.

RemoveCachedItem(TKey)

Removes an item from the cache.

```
public void RemoveCachedItem(TKey key)
```

Parameters

`key` TKey

The key of the item to remove.

TryGetValue(TKey, out T)

Attempts to retrieve a value from the cache.

```
public bool TryGetValue(TKey key, out T value)
```

Parameters

`key` TKey

The key to retrieve.

`value` T

When this method returns, contains the value associated with the key if found; otherwise, the default value for the type of the value parameter.

Returns

bool

`true` if the key exists in the cache; otherwise, `false`.

Struct ColliderEvent

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

Struct representing a collider event with callbacks for enter and exit interactions.

```
public struct ColliderEvent
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

OnEnter

The action to invoke when a collider with the matching tag enters this collider.

```
public Action<GameObject, ContactPoint[]> OnEnter
```

Field Value

Action<GameObject, ContactPoint[]>

Remarks

The first parameter is the other UnityEngine.GameObject involved in the collision. The second parameter is an array of UnityEngine.ContactPoint providing collision details.

OnExit

The action to invoke when a collider with the matching tag exits this collider.

```
public Action<GameObject, ContactPoint[]> OnExit
```

Field Value

Action<GameObject, ContactPoint[]>

Remarks

The first parameter is the other UnityEngine.GameObject involved in the collision. The second parameter is an array of UnityEngine.ContactPoint from the exit interaction.

Tag

The tag of the GameObject to match during collision events. Only GameObjects with this tag will trigger the callbacks.

```
public string Tag
```

Field Value

string

Struct ColliderEventEntry

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

Stores collision and trigger events.

```
public struct ColliderEventEntry
```

Inherited Members
ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Fields

CollisionEvent

```
public ColliderEvent CollisionEvent
```

Field Value

ColliderEvent

TriggerEvent

```
public ColliderEvent TriggerEvent
```

Field Value

ColliderEvent

Class ColliderEventHandler

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

Handles runtime registration of collision and trigger events based on object tags.

```
public abstract class ColliderEventHandler : MonoBehaviour
```

Inheritance

object ← ColliderEventHandler

Examples

```
public class MyColliderHandler : ColliderEventHandler
{
    private void Start()
    {
        // Register a collision event for objects tagged "Enemy"
        RegisterColliderEvent(new ColliderEvent
        {
            Tag = "Enemy",
            OnEnter = (go, contacts) => Debug.Log($"Enemy collided with {go.name}"),
            OnExit = (go, contacts) => Debug.Log($"Enemy stopped colliding with {go.name}")
        });

        // Register a trigger event for objects tagged "Pickup"
        RegisterColliderEvent(new ColliderEvent
        {
            Tag = "Pickup",
            OnEnter = (go, _) => Debug.Log($"Entered pickup trigger: {go.name}"),
            OnExit = (go, _) => Debug.Log($"Exited pickup trigger: {go.name}")
        }, isTriggerEvent: true);
    }
}
```

Methods

RegisterColliderEvent(ColliderEvent, bool)

Registers a collider event by tag.

```
protected void RegisterColliderEvent(ColliderEvent colliderEvent, bool isTriggerEvent
= false)
```

Parameters

`colliderEvent` ColliderEvent

`isTriggerEvent` bool

UnregisterColliderEvent(string)

Unregisters a collider event by a given tag.

```
protected void UnregisterColliderEvent(string eventTag)
```

Parameters

`eventTag` string

Class EphemeralSingleton<T>

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

An ephemeral singleton that does not persist across scene loads.

```
public abstract class EphemeralSingleton<T> : Singleton<T> where T : MonoBehaviour
```

Type Parameters

T

Type of the singleton class.

Inheritance

object ← Singleton<T> ← EphemeralSingleton<T>

Derived

CacheDictionary<TKey, T>, GenericObjectPool<T>

Inherited Members

Singleton<T>.instance , Singleton<T>.Instance , Singleton<T>.Awake()

Examples

```
// Example usage of EphemeralSingleton
public class UIManager : EphemeralSingleton<UIManager>
{
    public void ShowMenu()
    {
        Debug.Log("Showing menu");
    }
}

public class MainMenu : MonoBehaviour
{
    void OnEnable()
    {
        UIManager.Instance.ShowMenu();
```

```
    }
}
```

Class GenericObjectPool<T>

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

A generic object pool for managing reusable objects of type T.

```
public abstract class GenericObjectPool<T> : EphemeralSingleton<GenericObjectPool<T>> where
T : Component
```

Type Parameters

T

The type of objects to pool, which must be a Component.

Inheritance

object ← Singleton<GenericObjectPool<T>> ← EphemeralSingleton<GenericObjectPool<T>> ← GenericObjectPool<T>

Inherited Members

Singleton<GenericObjectPool<T>>.instance , Singleton<GenericObjectPool<T>>.Instance

Fields

activatePooledObjects

```
protected bool activatePooledObjects
```

Field Value

bool

activeObjects

```
protected readonly LinkedList<T> activeObjects
```

Field Value

LinkedList<T>

allObjects

```
protected readonly HashSet<T> allObjects
```

Field Value

HashSet<T>

allowObjectRecycling

```
protected bool allowObjectRecycling
```

Field Value

bool

awakeObjectsBatchSize

```
protected int awakeObjectsBatchSize
```

Field Value

int

awakeObjectsOnCreation

```
protected bool awakeObjectsOnCreation
```

Field Value

bool

maxPoolSize

```
protected int maxPoolSize
```

Field Value

int

prefab

```
protected T prefab
```

Field Value

T

prewarmCount

```
protected int prewarmCount
```

Field Value

int

readyToUseObjects

```
protected readonly Queue<T> readyToUseObjects
```

Field Value

Queue<T>

Properties

ActiveObjects

```
public IReadOnlyCollection<T> ActiveObjects { get; }
```

Property Value

IReadOnlyCollection<T>

The collection of currently active objects in the pool.

AvailableSlots

```
public int AvailableSlots { get; }
```

Property Value

int

The number of available slots in the pool for new objects.

Methods

AddObjects(int)

Adds a specified number of objects to the pool, up to the available slots.

```
protected virtual void AddObjects(int count)
```

Parameters

`count` int

The number of objects to add to the pool.

Awake()

Initializes the pool and prewarms objects if specified.

```
protected override void Awake()
```

Examples

```
public class Bullet : MonoBehaviour
{
    // Bullet logic here
}

public class BulletPool : GenericObjectPool<Bullet>
{
    // Optionally add pool-specific logic here
    // The Pool has to be placed in the scene. Within the inspector, you have to assign
it's prefab.
}

public class PlayerShooter : MonoBehaviour
{
    private void Shoot()
    {
        Bullet bullet = BulletPool.Instance.Get();
        bullet.transform.position = transform.position;
        bullet.gameObject.SetActive(true);

        // Setup bullet, e.g., velocity, direction...
    }

    private void OnBulletFinished(Bullet bullet)
    {
        BulletPool.Instance.ReturnToPool(bullet);
    }
}
```

CreateInstance()

Creates a new instance of the prefab and initializes it for use in the pool.

```
protected virtual T CreateInstance()
```

Returns

T

A new instance of type `T` with its GameObject deactivated.

Get()

Retrieves an object from the pool. If no objects are available, attempts to create or recycle one.

```
public virtual T Get()
```

Returns

T

A pooled object of type `T` or `null` if the operation fails.

Examples

```
// Example: shooting bullets
Bullet bullet = BulletPool.Instance.Get();
if (bullet != null)
{
    bullet.transform.position = transform.position;
}
```

Get(Vector3, Quaternion, Transform)

Retrieves an object from the pool and sets its position, rotation, and parent transform. If no objects are available, attempts to create or recycle one.

```
public virtual T Get(Vector3 position, Quaternion rotation, Transform parent = null)
```

Parameters

`position` Vector3

The position to set for the object.

`rotation` Quaternion

The rotation to set for the object.

`parent` Transform

The parent transform to assign to the object. Defaults to `null`.

Returns

T

A pooled object of type `T` or `null` if the operation fails.

Examples

```
Vector3 pos = transform.position + transform.forward * 2f;
Quaternion rot = Quaternion.identity;
Bullet bullet = BulletPool.Instance.Get(pos, rot, this.transform);
```

GetOrCreateObject(out bool)

Retrieves an object from the pool or creates/recycles one if necessary.

```
protected virtual T GetOrCreateObject(out bool wasNew)
```

Parameters

`wasNew` bool

Outputs whether the object was newly created (`true`) or retrieved from the pool (`false`).

Returns

T

A pooled object of type `T` or `null` if the operation fails.

NotifyPooled(T, bool)

Notifies that an object has been pooled and invokes relevant events.

```
protected virtual void NotifyPooled(T obj, bool wasNew)
```

Parameters

`obj` T

The object that has been pooled.

`wasNew` bool

Indicates whether the object was newly created (`true`) or retrieved from the pool (`false`).

PrewarmCoroutine(int)

Prewarms the object pool by activating and deactivating objects in batches. This ensures that objects are properly initialized before use.

```
protected virtual IEnumerator PrewarmCoroutine(int batchSize = 50)
```

Parameters

`batchSize` int

The number of objects to activate and deactivate in each batch. Defaults to 50.

Returns

IEnumerator

An enumerator that performs the prewarming operation over multiple frames.

ReturnToPool(T)

Returns an object to the pool.

```
public virtual void ReturnToPool(T objectToReturn)
```

Parameters

`objectToReturn` T

The object to return to the pool.

Examples

```
// When a bullet finishes:
BulletPool.Instance.ReturnToPool(bullet);
```

Events

OnObjectPooled

```
public event Action<T> OnObjectPooled
```

Event Type

Action<T>

OnObjectRecycled

```
public event Action<T> OnObjectRecycled
```

Event Type

Action<T>

OnObjectReturned

```
public event Action<T> OnObjectReturned
```

Event Type

Action<T>

OnPrewarmCompleted

```
public event Action<List<T>> OnPrewarmCompleted
```

Event Type

Action<List<T>>

Interface IPoolable

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

Interface for objects that can be managed by a pool. Provides methods to handle pooling, recycling, and returning operations.

```
public interface IPoolable
```

Methods

OnPooled()

```
void OnPooled()
```

OnRecycled()

```
void OnRecycled()
```

OnReturned()

```
void OnReturned()
```

Class PersistentObject

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

A Unity component that assigns a unique GUID to the object for persistent identification.

```
public class PersistentObject : MonoBehaviour
```

Inheritance

object ← PersistentObject

Examples

```
// Attach PersistentObject component to any GameObject in the editor.
// When you create or duplicate the GameObject, it automatically gets a unique GUID.

public class ExampleUsage : MonoBehaviour
{
    private void Start()
    {
        PersistentObject persistent = gameObject.GetComponent<PersistentObject>();
        if (persistent != null)
        {
            Debug.Log("Persistent GUID: " + persistent.guid);
        }
    }
}
```

Fields

guid

```
public string guid
```

Field Value

string

Class PersistentSingleton<T>

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

A persistent singleton that is not destroyed on scene load.

```
public abstract class PersistentSingleton<T> : Singleton<T> where T : MonoBehaviour
```

Type Parameters

T

Type of the singleton class.

Inheritance

object ← Singleton<T> ← PersistentSingleton<T>

Inherited Members

Singleton<T>.instance , Singleton<T>.Instance

Examples

```
// Example usage of PersistentSingleton
public class AudioManager : PersistentSingleton<AudioManager>
{
    public void PlaySound(string clipName)
    {
        Debug.Log("Playing sound: " + clipName);
    }
}

public class GameController : MonoBehaviour
{
    void Start()
    {
        AudioManager.Instance.PlaySound("GameStart");
    }
}
```

Methods

Awake()

Awake method to call DontDestroyOnLoad on the object.

```
protected override void Awake()
```

Class Singleton<T>

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

Abstract base class for creating singleton MonoBehaviour instances.

```
public abstract class Singleton<T> : MonoBehaviour where T : MonoBehaviour
```

Type Parameters

T

Type of the singleton class.

Inheritance

object ← Singleton<T>

Derived

EphemeralSingleton<T>, PersistentSingleton<T>

Fields

instance

```
protected static T instance
```

Field Value

T

Properties

Instance

Gets the singleton instance. If the instance is not found, it searches for it in the scene.

```
public static T Instance { get; }
```

Property Value

T

The singleton instance of type T.

Methods

Awake()

Awake method to initialize the singleton instance.

```
protected virtual void Awake()
```

Class TransformAnimator

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

A component that animates the position, rotation, and scale of a target Transform based on configurable animation settings.

```
public class TransformAnimator : MonoBehaviour
```

Inheritance

object ← TransformAnimator

Fields

isEnabled

```
protected bool isEnabled
```

Field Value

bool

isPositionForward

```
protected bool isPositionForward
```

Field Value

bool

isRotationForward

```
protected bool isRotationForward
```

Field Value

bool

isScaleForward

```
protected bool isScaleForward
```

Field Value

bool

positionConfig

```
protected TransformAnimator.TransformAnimatorComponentData positionConfig
```

Field Value

TransformAnimator.TransformAnimatorComponentData

positionDelayTimer

```
protected float positionDelayTimer
```

Field Value

float

positionProgress

```
protected float positionProgress
```

Field Value

float

positionTarget

```
protected Vector3 positionTarget
```

Field Value

Vector3

rotationConfig

```
protected TransformAnimator.TransformAnimatorComponentData rotationConfig
```

Field Value

TransformAnimator.TransformAnimatorComponentData

rotationDelayTimer

```
protected float rotationDelayTimer
```

Field Value

float

rotationProgress

```
protected float rotationProgress
```

Field Value

float

rotationTarget

```
protected Vector3 rotationTarget
```

Field Value

Vector3

scaleConfig

```
protected TransformAnimator.TransformAnimatorComponentData scaleConfig
```

Field Value

TransformAnimator.TransformAnimatorComponentData

scaleDelayTimer

```
protected float scaleDelayTimer
```

Field Value

float

scaleProgress

```
protected float scaleProgress
```

Field Value

float

scaleTarget

```
protected Vector3 scaleTarget
```

Field Value

Vector3

target

```
protected Transform target
```

Field Value

Transform

Methods

AnimatePosition()

Animates the position of the target Transform based on the configuration.

```
protected virtual void AnimatePosition()
```

AnimateRotation()

Animates the rotation of the target Transform based on the configuration.

```
protected virtual void AnimateRotation()
```

AnimateScale()

Animates the scale of the target Transform based on the configuration.

```
protected virtual void AnimateScale()
```

Awake()

Initializes the target Transform and sets its initial position, rotation, and scale based on the animation configurations. Also calculates randomized targets for animations.

```
protected virtual void Awake()
```

GetRandomizedTarget(Vector3, Vector3)

Calculates a randomized target value by applying a random offset to the base end value.

```
protected static Vector3 GetRandomizedTarget(Vector3 baseEnd, Vector3 offset)
```

Parameters

`baseEnd` Vector3

The base end value to which the random offset will be applied.

`offset` Vector3

The range of random offset to apply to each axis.

Returns

Vector3

A new Vector3 with randomized values within the specified offset range.

SetIsEnabled(bool)

Enables or disables the animator.

```
public virtual void SetIsEnabled(bool state)
```

Parameters

`state` bool

True to enable, false to disable.

Update()

Updates the animation for position, rotation, and scale on each frame.

```
protected virtual void Update()
```

Class TransformAnimator.TransformAnimatorComponentData

Namespace: UniUtils.GameObjects

Assembly: cs.temp.dll.dll

Represents the configuration data for animating a specific transform component.

```
public class TransformAnimator.TransformAnimatorComponentData
```

Inheritance

object ← TransformAnimator.TransformAnimatorComponentData

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

animationCurve

```
public AnimationCurve animationCurve
```

Field Value

AnimationCurve

canLoop

```
public bool canLoop
```

Field Value

bool

endValue

```
public Vector3 endValue
```

Field Value

Vector3

isEnabled

```
public bool isEnabled
```

Field Value

bool

loopDelay

```
public float loopDelay
```

Field Value

float

speed

```
public float speed
```

Field Value

float

startValue

```
public Vector3 startValue
```

Field Value

Vector3

targetVariance

```
public Vector3 targetVariance
```

Field Value

Vector3

Namespace UniUtils.Reflection

Classes

PredefinedAssemblyUtil

Utility class for retrieving types from predefined assemblies.

Class PredefinedAssemblyUtil

Namespace: UniUtils.Reflection

Assembly: cs.temp.dll.dll

Utility class for retrieving types from predefined assemblies.

```
public static class PredefinedAssemblyUtil
```

Inheritance

object ← PredefinedAssemblyUtil

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetTypes(Type, bool)

Gets a list of types that are subclasses of the specified generic base type.

```
public static List<Type> GetTypes(Type genericBaseType, bool includeInterfaces = false)
```

Parameters

`genericBaseType` Type

The generic base type to match.

`includeInterfaces` bool

Whether to include interfaces in the search.

Returns

List<Type>

A list of types that are subclasses of the specified generic base type.

Examples

Example usage:

```
List<Type> derivedTypes = TypeUtility.GetTypes(typeof(MyGenericBase<>));
foreach (Type t in derivedTypes)
{
    Debug.Log($"Found: {t.FullName}");
}
```

Exceptions

System.Reflection.ReflectionTypeLoadException

Thrown when there is an error loading types from an assembly.

Namespace UniUtils.SceneManagement

Classes

SceneLoader

Manages scene loading operations in Unity.

Class SceneLoader

Namespace: UniUtils.SceneManagement

Assembly: cs.temp.dll.dll

Manages scene loading operations in Unity.

```
public class SceneLoader : PersistentSingleton<SceneLoader>
```

Inheritance

object ← SceneLoader

Examples

```
// Example usage of SceneLoader
public class GameFlowManager : MonoBehaviour
{
    private void Start()
    {
        // Subscribe to scene loading events
        SceneLoader.Instance.OnSceneChangeStart += OnSceneStart;
        SceneLoader.Instance.OnSceneChangeProgressUpdate += OnProgressUpdate;
        SceneLoader.Instance.OnSceneChangeFinished += OnSceneFinished;

        // Start loading scene with index 2
        SceneLoader.Instance.LoadScene(2, () => Debug.Log("Scene loaded successfully!"));
    }

    private void OnSceneStart(int sceneIndex)
    {
        Debug.Log("Loading started for scene: " + sceneIndex);
    }

    private void OnProgressUpdate(float progress)
    {
        Debug.Log("Loading progress: " + (progress * 100) + "%");
    }

    private void OnSceneFinished(int sceneIndex)
    {
        Debug.Log("Loading finished for scene: " + sceneIndex);
```

```
    }
}
```

Methods

LoadScene(int, Action)

Starts the asynchronous scene loading process.

```
public Coroutine LoadScene(int sceneIndex, Action callback = null)
```

Parameters

`sceneIndex` int

The index of the scene to load.

`callback` Action

Optional callback to invoke after the scene is loaded.

Returns

Coroutine

A Coroutine for the asynchronous operation.

Events

OnSceneChangeFinished

Event triggered when a scene change finishes.

```
public event Action<int> OnSceneChangeFinished
```

Event Type

Action<int>

OnSceneChangeProgressUpdate

Event triggered to update the progress of a scene change.

```
public event Action<float> OnSceneChangeProgressUpdate
```

Event Type

Action<float>

OnSceneChangeStart

Event triggered when a scene change starts.

```
public event Action<int> OnSceneChangeStart
```

Event Type

Action<int>