

Concurrent IO

Generated by Doxygen 1.14.0

1 Namespace Index	1
1.1 Namespace List	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Namespace Documentation	7
4.1 ProjectGamedev Namespace Reference	7
4.2 ProjectGamedev.IO Namespace Reference	7
4.3 ProjectGamedev.IO.Tests Namespace Reference	7
4.4 ProjectGamedev.IO.Tests.PlayModeTests Namespace Reference	8
5 Class Documentation	9
5.1 ProjectGamedev.IO.ConcurrentIO Class Reference	9
5.1.1 Detailed Description	10
5.1.2 Member Function Documentation	10
5.1.2.1 CriticalSection() [1/2]	10
5.1.2.2 CriticalSection() [2/2]	10
5.1.2.3 ExclusiveSection()	11
5.1.2.4 PruneLocks()	11
5.1.3 Member Data Documentation	11
5.1.3.1 locks	11
5.1.3.2 MAX_LOCK_COUNT	12
5.1.3.3 readerWriterLock	12
5.2 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest Class Reference	12
5.2.1 Detailed Description	13
5.2.2 Member Function Documentation	13
5.2.2.1 BeforeAll()	13
5.2.2.2 CopyDirectory()	13
5.2.2.3 TestConcurrentAccessToFile()	13
5.2.2.4 TestCriticalSectionParameterChecks()	13
5.2.2.5 TestDualKeyLocking()	14
5.2.2.6 TestExclusiveLocking()	14
5.2.2.7 TestExclusiveLockingNoTimeout()	14
5.2.2.8 TestExclusiveSectionParameterChecks()	14
5.2.3 Member Data Documentation	14
5.2.3.1 testdirectorypath1	14
5.2.3.2 testdirectorypath2	14
5.2.3.3 testfilepath1	14
5.2.3.4 testfilepath2	15
5.2.3.5 testfilepath3	15

5.3 ProjectGamedev.IO.Directory Class Reference	15
5.3.1 Detailed Description	17
5.3.2 Member Function Documentation	17
5.3.2.1 CreateDirectory()	17
5.3.2.2 Delete() [1/2]	18
5.3.2.3 Delete() [2/2]	19
5.3.2.4 EnumerateDirectories() [1/4]	20
5.3.2.5 EnumerateDirectories() [2/4]	21
5.3.2.6 EnumerateDirectories() [3/4]	22
5.3.2.7 EnumerateDirectories() [4/4]	24
5.3.2.8 EnumerateFiles() [1/4]	25
5.3.2.9 EnumerateFiles() [2/4]	26
5.3.2.10 EnumerateFiles() [3/4]	27
5.3.2.11 EnumerateFiles() [4/4]	29
5.3.2.12 EnumerateFileSystemEntries() [1/4]	30
5.3.2.13 EnumerateFileSystemEntries() [2/4]	31
5.3.2.14 EnumerateFileSystemEntries() [3/4]	32
5.3.2.15 EnumerateFileSystemEntries() [4/4]	34
5.3.2.16 Exists()	35
5.3.2.17 GetCreationTime()	36
5.3.2.18 GetCreationTimeUtc()	37
5.3.2.19 GetCurrentDirectory()	38
5.3.2.20 GetDirectories() [1/4]	38
5.3.2.21 GetDirectories() [2/4]	39
5.3.2.22 GetDirectories() [3/4]	40
5.3.2.23 GetDirectories() [4/4]	42
5.3.2.24 GetDirectoryRoot()	43
5.3.2.25 GetFiles() [1/4]	44
5.3.2.26 GetFiles() [2/4]	45
5.3.2.27 GetFiles() [3/4]	46
5.3.2.28 GetFiles() [4/4]	48
5.3.2.29 GetFileSystemEntries() [1/4]	49
5.3.2.30 GetFileSystemEntries() [2/4]	50
5.3.2.31 GetFileSystemEntries() [3/4]	51
5.3.2.32 GetFileSystemEntries() [4/4]	53
5.3.2.33 GetLastAccessTime()	54
5.3.2.34 GetLastAccessTimeUtc()	55
5.3.2.35 GetLastWriteTime()	56
5.3.2.36 GetLastWriteTimeUtc()	57
5.3.2.37 GetLogicalDrives()	58
5.3.2.38 GetParent()	58
5.3.2.39 Move()	59

5.3.2.40 ReturningExclusiveSection< T >()	60
5.3.2.41 SetCreationTime()	60
5.3.2.42 SetCreationTimeUtc()	61
5.3.2.43 SetCurrentDirectory()	62
5.3.2.44 SetLastAccessTime()	62
5.3.2.45 SetLastAccessTimeUtc()	63
5.3.2.46 SetLastWriteTime()	64
5.3.2.47 SetLastWriteTimeUtc()	65
5.4 ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest Class Reference	66
5.4.1 Detailed Description	66
5.4.2 Member Function Documentation	66
5.4.2.1 BeforeAll()	66
5.4.2.2 CallAllDirectoryMethods()	66
5.4.2.3 TestDirectoryMethodsNoTimeoutsNoExceptions()	66
5.4.3 Member Data Documentation	67
5.4.3.1 testdirectorypath1	67
5.4.3.2 testdirectorypath2	67
5.4.3.3 testfilepath1	67
5.4.3.4 testfilepath2	67
5.5 ProjectGamedev.IO.File Class Reference	67
5.5.1 Detailed Description	69
5.5.2 Member Function Documentation	69
5.5.2.1 AppendAllLines() [1/2]	69
5.5.2.2 AppendAllLines() [2/2]	70
5.5.2.3 AppendAllText() [1/2]	71
5.5.2.4 AppendAllText() [2/2]	72
5.5.2.5 Copy() [1/2]	72
5.5.2.6 Copy() [2/2]	73
5.5.2.7 Create() [1/3]	74
5.5.2.8 Create() [2/3]	75
5.5.2.9 Create() [3/3]	76
5.5.2.10 Delete()	77
5.5.2.11 Exists()	77
5.5.2.12 GetAttributes()	78
5.5.2.13 GetCreationTime()	79
5.5.2.14 GetCreationTimeUtc()	80
5.5.2.15 GetLastAccessTime()	80
5.5.2.16 GetLastAccessTimeUtc()	81
5.5.2.17 GetLastWriteTime()	82
5.5.2.18 GetLastWriteTimeUtc()	83
5.5.2.19 Move()	84
5.5.2.20 ReadAllBytes()	85

5.5.2.21 ReadAllLines() [1/2]	86
5.5.2.22 ReadAllLines() [2/2]	87
5.5.2.23 ReadAllText() [1/2]	87
5.5.2.24 ReadAllText() [2/2]	88
5.5.2.25 Replace() [1/2]	89
5.5.2.26 Replace() [2/2]	90
5.5.2.27 ReturningCriticalSection< T >()	91
5.5.2.28 SetAttributes()	91
5.5.2.29 SetCreationTime()	92
5.5.2.30 SetCreationTimeUtc()	93
5.5.2.31 SetLastAccessTime()	94
5.5.2.32 SetLastAccessTimeUtc()	95
5.5.2.33 SetLastWriteTime()	95
5.5.2.34 SetLastWriteTimeUtc()	96
5.5.2.35 WriteAllBytes()	96
5.5.2.36 WriteAllLines() [1/4]	97
5.5.2.37 WriteAllLines() [2/4]	98
5.5.2.38 WriteAllLines() [3/4]	99
5.5.2.39 WriteAllLines() [4/4]	100
5.5.2.40 WriteAllText() [1/2]	100
5.5.2.41 WriteAllText() [2/2]	101
5.6 ProjectGamedev.IO.Tests.PlayModeTests.FileTest Class Reference	102
5.6.1 Detailed Description	102
5.6.2 Member Function Documentation	103
5.6.2.1 BeforeAll()	103
5.6.2.2 CallAllFileMethods()	103
5.6.2.3 TestFileMethodsNoTimeoutsNoExceptions()	103
5.6.3 Member Data Documentation	103
5.6.3.1 testdirectorypath1	103
5.6.3.2 testdirectorypath2	103
5.6.3.3 testfilepath1	103
5.6.3.4 testfilepath2	103
5.6.3.5 testfilepath3	103
5.6.3.6 testfilepath4	104
5.7 ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings Class Reference	104
5.7.1 Detailed Description	104
5.7.2 Member Function Documentation	104
5.7.2.1 GlobalSetUp()	104
5.7.2.2 GlobalTeardown()	104
5.8 ProjectGamedev.IO.ConcurrentIO.LockObject Class Reference	104
5.8.1 Detailed Description	105
5.9 ProjectGamedev.IO.Tests.TestConstants Class Reference	105

5.9.1 Detailed Description	105
5.9.2 Member Data Documentation	105
5.9.2.1 TEST_DIRECTORY_NAME_1	105
5.9.2.2 TEST_DIRECTORY_NAME_2	106
5.9.2.3 TEST_DIRECTORY_NAME_3	106
5.9.2.4 TEST_DIRECTORY_NAME_4	106
5.9.2.5 TEST_DIRECTORY_NAME_5	106
5.9.2.6 TEST_DIRECTORY_NAME_6	106
5.9.2.7 TEST_FILE_NAME_1	106
5.9.2.8 TEST_FILE_NAME_2	106
5.9.2.9 TEST_FILE_NAME_3	106
5.9.2.10 TEST_FILE_NAME_4	106
5.9.2.11 TEST_FILE_NAME_5	107
5.9.2.12 TEST_FILE_NAME_6	107
5.9.2.13 TEST_FILE_NAME_7	107
5.9.2.14 TEST_FILE_NAME_8	107
5.9.2.15 TEST_FILE_NAME_9	107
5.9.2.16 TEST_ROOT_FOLDER	107
5.9.2.17 TEST_THREAD_COUNT	107
5.9.2.18 TIMEOUT_LARGE_MILLISECONDS	107
5.9.2.19 TIMEOUT_MILLISECONDS	107
5.10 ProjectGameDev.IO.Tests.TestFactory Class Reference	108
5.10.1 Detailed Description	108
5.10.2 Member Function Documentation	108
5.10.2.1 CreateTestDirectory()	108
5.10.2.2 CreateTestFile()	108
5.10.2.3 DeleteTestRootDirectory()	109
5.10.3 Member Data Documentation	109
5.10.3.1 Root	109
6 File Documentation	111
6.1 Assets/Project GAMEDEV/Tools/IO/Scripts/ConcurrentIO.cs File Reference	111
6.2 Assets/Project GAMEDEV/Tools/IO/Scripts/Directory.cs File Reference	111
6.3 Assets/Project GAMEDEV/Tools/IO/Scripts/File.cs File Reference	112
6.4 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/ConcurrentIOTest.cs File Reference	112
6.5 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/DirectoryTest.cs File Reference	112
6.6 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/FileTest.cs File Reference	113
6.7 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/GlobalTestSettings.cs File Reference	113
6.8 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/TestConstants.cs File Reference	113
6.9 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/TestFactory.cs File Reference	114
Index	115

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ProjectGamedev	7
ProjectGamedev.IO	7
ProjectGamedev.IO.Tests	7
ProjectGamedev.IO.Tests.PlayModeTests	8

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[ProjectGamedev.IO.ConcurrentIO](#)

A simple concurrent wrapper for performing [IO](#) operations in parallel.

This class uses named mutex locks to ensure 2 threads do not simultaneously operate on the same file path,

and to also allow several threads to simultaneously operate on different file or directory paths 9

[ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest](#)

Test class containing test cases for the [ConcurrentIO](#) class 12

[ProjectGamedev.IO.Directory](#)

A lightweight thread-safe directory utility serving as an abstraction over the raw [System.IO.Directory](#) class.

Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#).

Does not handle exceptions 15

[ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest](#)

Test class containing test cases for the [Directory](#) class 66

[ProjectGamedev.IO.File](#)

A lightweight thread-safe file utility serving as an abstraction over the raw [System.IO.File](#) class.

Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#).

Does not handle exceptions 67

[ProjectGamedev.IO.Tests.PlayModeTests.FileTest](#)

Test class containing test cases for the [File](#) class 102

[ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings](#)

SetUpFixture class for the tests. Defines global pre- and post-test setup and teardown 104

[ProjectGamedev.IO.ConcurrentIO.LockObject](#)

An empty class. Lock objects created from this class can be used with C#'s `lock` keyword for locking 104

[ProjectGamedev.IO.Tests.TestConstants](#)

Container for constants used throughout the tests 105

[ProjectGamedev.IO.Tests.TestFactory](#)

This class is used in test setup phases (BeforeAll/AfterAll/etc) to create and delete the necessary test data (e.g. directories and files) 108

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Assets/Project GAMEDEV/Tools/IO/Scripts/ ConcurrentIO.cs	111
Assets/Project GAMEDEV/Tools/IO/Scripts/ Directory.cs	111
Assets/Project GAMEDEV/Tools/IO/Scripts/ File.cs	112
Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/ TestConstants.cs	113
Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/ TestFactory.cs	114
Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/ ConcurrentIOTest.cs	112
Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/ DirectoryTest.cs	112
Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/ FileTest.cs	113
Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/ GlobalTestSettings.cs	113

Chapter 4

Namespace Documentation

4.1 ProjectGamedev Namespace Reference

Namespaces

- namespace [IO](#)

4.2 ProjectGamedev.IO Namespace Reference

Namespaces

- namespace [Tests](#)

Classes

- class [ConcurrentIO](#)
*A simple concurrent wrapper for performing [IO](#) operations in parallel.
This class uses named mutex locks to ensure 2 threads do not simultaneously operate on the same file path, and to also allow several threads to simultaneously operate on different file or directory paths.*
- class [Directory](#)
A lightweight thread-safe directory utility serving as an abstraction over the raw `System.IO.Directory` class. Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#). Does not handle exceptions.
- class [File](#)
A lightweight thread-safe file utility serving as an abstraction over the raw `System.IO.File` class. Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#). Does not handle exceptions.

4.3 ProjectGamedev.IO.Tests Namespace Reference

Namespaces

- namespace [PlayModeTests](#)

Classes

- class [TestConstants](#)
Container for constants used throughout the tests.
- class [TestFactory](#)
This class is used in test setup phases (BeforeAll/AfterAll/etc) to create and delete the necessary test data (e.g. directories and files).

4.4 ProjectGamedev.IO.Tests.PlayModeTests Namespace Reference

Classes

- class [ConcurrentIOTest](#)
Test class containing test cases for the [ConcurrentIO](#) class.
- class [DirectoryTest](#)
Test class containing test cases for the [Directory](#) class.
- class [FileTest](#)
Test class containing test cases for the [File](#) class.
- class [GlobalTestSettings](#)
SetUpFixture class for the tests. Defines global pre- and post-test setup and teardown.

Chapter 5

Class Documentation

5.1 ProjectGamedev.IO.ConcurrentIO Class Reference

A simple concurrent wrapper for performing [IO](#) operations in parallel.

This class uses named mutex locks to ensure 2 threads do not simultaneously operate on the same file path, and to also allow several threads to simultaneously operate on different file or directory paths.

Classes

- class [LockObject](#)

An empty class. Lock objects created from this class can be used with C#'s `lock` keyword for locking.

Static Public Member Functions

- static void [CriticalSection](#) (string key, Action action)
Uses a named mutex (exclusive) lock to ensure action is executed in a thread-safe manner. Actions with the same key will run sequentially, but actions with different key can run in parallel.
- static void [CriticalSection](#) (string key1, string key2, Action action)
Uses 2 named mutex (exclusive) locks to ensure action concerning 2 paths (e.g. copying of files) is executed in a thread-safe manner. Actions with the same key1 OR key2 will run sequentially, but actions with different key1 AND key2 can run in parallel.
- static void [ExclusiveSection](#) (Action action)
Uses an exclusive lock to ensure action is executed in a thread-safe manner. Unlike `CriticalSection(string, System.Action)` and `CriticalSection(string, string, System.Action)`, this method uses an exclusive lock that halts all other methods in this class until released. Use this method for operations that require full exclusive access and do not allow any other operation to perform simultaneously (e.g. directory manipulation).

Static Private Member Functions

- static void [PruneLocks](#) ()
Clears the lock collection if the number of named locks exceeds the max allowed amount. Makes use of the exclusive locking mechanism to do so in a thread-safe manner.

Static Private Attributes

- const int `MAX_LOCK_COUNT` = 3
*Maximum allowed number of named locks before the collection is cleared.
This ensures the collection of locks does not explode in size during long application runs.*
- static readonly ConcurrentDictionary< string, LockObject > `locks` = new()
A thread-safe dictionary collection that holds a lock object for a specific string path. Periodically pruned.
- static readonly ReaderWriterLockSlim `readerWriterLock` = new(LockRecursionPolicy.NoRecursion)
Used to allow many `CriticalSection()` threads ("readers"), but only 1 `ExclusiveSection()` thread ("writer"). In other words, many threads can execute `CriticalSection()`, but once a thread tries to execute `ExclusiveSection()`, it will wait for the critical sections, then execute the exclusive section, and prevent any critical sections from starting until the exclusive section is finished.

5.1.1 Detailed Description

A simple concurrent wrapper for performing IO operations in parallel.

This class uses named mutex locks to ensure 2 threads do not simultaneously operate on the same file path, and to also allow several threads to simultaneously operate on different file or directory paths.

5.1.2 Member Function Documentation

5.1.2.1 CriticalSection() [1/2]

```
void ProjectGamedev.IO.ConcurrentIO.CriticalSection (
    string key,
    Action action) [inline], [static]
```

Uses a named mutex (exclusive) lock to ensure *action* is executed in a thread-safe manner. Actions with the same *key* will run sequentially, but actions with different *key* can run in parallel.

Parameters

<i>key</i>	This action's key (in the case of IO, file path). 2 actions with the same key will never run simultaneously, but 2 actions with different keys can.
<i>action</i>	The callable action to execute in a thread-safe manner.

Exceptions

<i>ArgumentException</i>	If <i>key</i> is an empty string.
<i>ArgumentNullException</i>	If any parameter is null.

5.1.2.2 CriticalSection() [2/2]

```
void ProjectGamedev.IO.ConcurrentIO.CriticalSection (
    string key1,
    string key2,
    Action action) [inline], [static]
```

Uses 2 named mutex (exclusive) locks to ensure *action* concerning 2 paths (e.g. copying of files) is executed in a thread-safe manner.

Actions with the same *key1* OR *key2* will run sequentially, but actions with different *key1* AND *key2* can run in parallel.

Parameters

<i>key1</i>	This action's first key (in the case of IO , file path).
<i>key2</i>	This action's second key (in the case of IO , file path).
<i>action</i>	The callable action to execute in a thread-safe manner.

Exceptions

<i>ArgumentException</i>	If any key is an empty string, or if keys are equal.
<i>ArgumentNullException</i>	If any parameter is <code>null</code> .

5.1.2.3 ExclusiveSection()

```
void ProjectGamedev.IO.ConcurrentIO.ExclusiveSection (
    Action action) [inline], [static]
```

Uses an exclusive lock to ensure *action* is executed in a thread-safe manner.

Unlike `CriticalSection(string, System.Action)` and `CriticalSection(string, string, System.Action)`, this method uses an exclusive lock that halts all other methods in this class until released.

Use this method for operations that require full exclusive access and do not allow any other operation to perform simultaneously (e.g. directory manipulation).

Parameters

<i>action</i>	The callable action to execute in a thread-safe manner.
---------------	---

Exceptions

<i>ArgumentNullException</i>	If <i>action</i> is <code>null</code> .
------------------------------	---

5.1.2.4 PruneLocks()

```
void ProjectGamedev.IO.ConcurrentIO.PruneLocks () [inline], [static], [private]
```

Clears the lock collection if the number of named locks exceeds the max allowed amount. Makes use of the exclusive locking mechanism to do so in a thread-safe manner.

5.1.3 Member Data Documentation

5.1.3.1 locks

```
readonly ConcurrentDictionary<string, LockObject> ProjectGamedev.IO.ConcurrentIO.locks = new()
[static], [private]
```

A thread-safe dictionary collection that holds a lock object for a specific string path. Periodically pruned.

5.1.3.2 MAX_LOCK_COUNT

```
const int ProjectGamedev.IO.ConcurrentIO.MAX_LOCK_COUNT = 3 [static], [private]
```

Maximum allowed number of named locks before the collection is cleared.
This ensures the collection of locks does not explode in size during long application runs.

5.1.3.3 readerWriterLock

```
readonly ReaderWriterLockSlim ProjectGamedev.IO.ConcurrentIO.readerWriterLock = new(Lock↔  
RecursionPolicy.NoRecursion) [static], [private]
```

Used to allow many [CriticalSection\(\)](#) threads ("readers"), but only 1 [ExclusiveSection\(\)](#) thread ("writer"). In other words, many threads can execute [CriticalSection\(\)](#), but once a thread tries to execute [ExclusiveSection\(\)](#), it will wait for the critical sections, then execute the exclusive section, and prevent any critical sections from starting until the exclusive section is finished.

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/[ConcurrentIO.cs](#)

5.2 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest Class Reference

Test class containing test cases for the [ConcurrentIO](#) class.

Public Member Functions

- void [BeforeAll](#) ()
- void [TestConcurrentAccessToFile](#) ()
Test ID: IO-3.
- void [TestDualKeyLocking](#) ()
Test ID: IO-4.
- void [TestExclusiveLocking](#) ()
Test ID: IO-5.
- void [TestExclusiveLockingNoTimeout](#) ()
Test ID: IO-6.
- void [TestCriticalSectionParameterChecks](#) ()
Test ID: IO-7.
- void [TestExclusiveSectionParameterChecks](#) ()

Static Private Member Functions

- static void [CopyDirectory](#) (string sourceDir, string destinationDir, bool recursive)

Private Attributes

- string [testfilepath1](#)
- string [testfilepath2](#)
- string [testfilepath3](#)
- string [testdirectorypath1](#)
- string [testdirectorypath2](#)

5.2.1 Detailed Description

Test class containing test cases for the [ConcurrentIO](#) class.

5.2.2 Member Function Documentation

5.2.2.1 BeforeAll()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.BeforeAll () [inline]
```

5.2.2.2 CopyDirectory()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.CopyDirectory (  
    string sourceDir,  
    string destinationDir,  
    bool recursive) [inline], [static], [private]
```

5.2.2.3 TestConcurrentAccessToFile()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.TestConcurrentAccessToFile ()  
[inline]
```

Test ID: IO-3.

[IO](#) operations concerning 1 file path are thread-safe.

5.2.2.4 TestCriticalSectionParameterChecks()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.TestCriticalSectionParameterChecks  
() [inline]
```

Test ID: IO-7.

[ConcurrentIO](#) checks parameters for validity. In case of invalid parameters, errors are thrown (no silent exit allowed!).

5.2.2.5 TestDualKeyLocking()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.TestDualKeyLocking () [inline]
```

Test ID: IO-4.

IO operations concerning 2 file paths are thread-safe.

5.2.2.6 TestExclusiveLocking()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.TestExclusiveLocking () [inline]
```

Test ID: IO-5.

IO operations concerning many directory path are thread-safe.

5.2.2.7 TestExclusiveLockingNoTimeout()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.TestExclusiveLockingNoTimeout ()  
[inline]
```

Test ID: IO-6.

Calling the methods of [ConcurrentIO](#) together in many threads does not cause deadlocks or timeouts.

5.2.2.8 TestExclusiveSectionParameterChecks()

```
void ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.TestExclusiveSectionParameter↵  
Checks () [inline]
```

5.2.3 Member Data Documentation

5.2.3.1 testdirectorypath1

```
string ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.testdirectorypath1 [private]
```

5.2.3.2 testdirectorypath2

```
string ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.testdirectorypath2 [private]
```

5.2.3.3 testfilepath1

```
string ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.testfilepath1 [private]
```

5.2.3.4 testfilepath2

```
string ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.testfilepath2 [private]
```

5.2.3.5 testfilepath3

```
string ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest.testfilepath3 [private]
```

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/[ConcurrentIOTest.cs](#)

5.3 ProjectGamedev.IO.Directory Class Reference

A lightweight thread-safe directory utility serving as an abstraction over the raw System.IO.Directory class. Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#). Does not handle exceptions.

Static Public Member Functions

- static DirectoryInfo [CreateDirectory](#) (string path)
Creates all directories and subdirectories in the specified path unless they already exist.
- static void [Delete](#) (string path)
Deletes an empty directory from a specified path.
- static void [Delete](#) (string path, bool recursive)
Deletes the specified directory and, if indicated, any subdirectories and files in the directory.
- static IEnumerable< string > [EnumerateDirectories](#) (string path)
Returns an enumerable collection of directory full names in a specified path.
- static IEnumerable< string > [EnumerateDirectories](#) (string path, string searchPattern)
Returns an enumerable collection of directory full names that match a search pattern in a specified path.
- static IEnumerable< string > [EnumerateDirectories](#) (string path, string searchPattern, EnumerationOptions enumerationOptions)
Returns an enumerable collection of the directory full names that match a search pattern in a specified path, and optionally searches subdirectories.
- static IEnumerable< string > [EnumerateDirectories](#) (string path, string searchPattern, SearchOption searchOption)
Returns an enumerable collection of directory full names that match a search pattern in a specified path, and optionally searches subdirectories.
- static IEnumerable< string > [EnumerateFiles](#) (string path)
Returns an enumerable collection of full file names in a specified path.
- static IEnumerable< string > [EnumerateFiles](#) (string path, string searchPattern)
Returns an enumerable collection of full file names that match a search pattern in a specified path.
- static IEnumerable< string > [EnumerateFiles](#) (string path, string searchPattern, SearchOption searchOption)
Returns an enumerable collection of full file names that match a search pattern in a specified path, and optionally searches subdirectories.
- static IEnumerable< string > [EnumerateFiles](#) (string path, string searchPattern, EnumerationOptions enumerationOptions)
Returns an enumerable collection of full file names that match a search pattern and enumeration options in a specified path, and optionally searches subdirectories.

- static IEnumerable< string > [EnumerateFileSystemEntries](#) (string path)
Returns an enumerable collection of file names and directory names in a specified path.
- static IEnumerable< string > [EnumerateFileSystemEntries](#) (string path, string searchPattern)
Returns an enumerable collection of file names and directory names that match a search pattern in a specified path.
- static IEnumerable< string > [EnumerateFileSystemEntries](#) (string path, string searchPattern, Enumeration↔ Options enumerationOptions)
Returns an enumerable collection of file names and directory names that match a search pattern and enumeration options in a specified path.
- static IEnumerable< string > [EnumerateFileSystemEntries](#) (string path, string searchPattern, SearchOption searchOption)
Returns an enumerable collection of file names and directory names that match a search pattern in a specified path, and optionally searches subdirectories.
- static bool [Exists](#) (string path)
Determines whether the given path refers to an existing directory on disk.
- static DateTime [GetCreationTime](#) (string path)
Gets the creation date and time of a directory.
- static DateTime [GetCreationTimeUtc](#) (string path)
Gets the creation date and time, in Coordinated Universal Time (UTC) format, of a directory.
- static string [GetCurrentDirectory](#) ()
Gets the current working directory of the application.
- static string[] [GetDirectories](#) (string path)
Returns the names of subdirectories (including their paths) in the specified directory.
- static string[] [GetDirectories](#) (string path, string searchPattern)
Returns the names of subdirectories (including their paths) that match the specified search pattern in the specified directory.
- static string[] [GetDirectories](#) (string path, string searchPattern, SearchOption searchOption)
Returns the names of the subdirectories (including their paths) that match the specified search pattern in the specified directory, and optionally searches subdirectories.
- static string[] [GetDirectories](#) (string path, string searchPattern, EnumerationOptions enumerationOptions)
Returns the names of subdirectories (including their paths) that match the specified search pattern and enumeration options in the specified directory.
- static string [GetDirectoryRoot](#) (string path)
Returns the volume information, root information, or both for the specified path.
- static string[] [GetFiles](#) (string path)
Returns the names of files (including their paths) in the specified directory.
- static string[] [GetFiles](#) (string path, string searchPattern)
Returns the names of files (including their paths) that match the specified search pattern in the specified directory.
- static string[] [GetFiles](#) (string path, string searchPattern, SearchOption searchOption)
Returns the names of files (including their paths) that match the specified search pattern in the specified directory, using a value to determine whether to search subdirectories.
- static string[] [GetFiles](#) (string path, string searchPattern, EnumerationOptions enumerationOptions)
Returns the names of files (including their paths) that match the specified search pattern and enumeration options in the specified directory.
- static string[] [GetFileSystemEntries](#) (string path)
Returns the names of all files and subdirectories in a specified path.
- static string[] [GetFileSystemEntries](#) (string path, string searchPattern)
Returns an array of file names and directory names that match a search pattern in a specified path.
- static string[] [GetFileSystemEntries](#) (string path, string searchPattern, EnumerationOptions enumeration↔ Options)
Returns an array of file names and directory names that match a search pattern and enumeration options in a specified path.
- static string[] [GetFileSystemEntries](#) (string path, string searchPattern, SearchOption searchOption)

Returns an array of all the file names and directory names that match a search pattern in a specified path, and optionally searches subdirectories.

- static DateTime [GetLastAccessTime](#) (string path)
Returns the date and time the specified file or directory was last accessed.
- static DateTime [GetLastAccessTimeUtc](#) (string path)
Returns the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last accessed.
- static DateTime [GetLastWriteTime](#) (string path)
Returns the date and time the specified file or directory was last written to.
- static DateTime [GetLastWriteTimeUtc](#) (string path)
Returns the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last written to.
- static string[] [GetLogicalDrives](#) ()
Retrieves the names of the logical drives on this computer in the form "<drive letter>:\".
- static DirectoryInfo [GetParent](#) (string path)
Retrieves the parent directory of the specified path, including both absolute and relative paths.
- static void [Move](#) (string sourceDirName, string destDirName)
Moves a file or a directory and its contents to a new location.
- static void [SetCreationTime](#) (string path, DateTime creationTime)
Sets the creation date and time for the specified file or directory.
- static void [SetCreationTimeUtc](#) (string path, DateTime creationTimeUtc)
Sets the creation date and time, in Coordinated Universal Time (UTC) format, for the specified file or directory.
- static void [SetCurrentDirectory](#) (string path)
Sets the application's current working directory to the specified directory.
- static void [SetLastAccessTime](#) (string path, DateTime lastAccessTime)
Sets the date and time the specified file or directory was last accessed.
- static void [SetLastAccessTimeUtc](#) (string path, DateTime lastAccessTimeUtc)
Sets the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last accessed.
- static void [SetLastWriteTime](#) (string path, DateTime lastWriteTime)
Sets the date and time a directory was last written to.
- static void [SetLastWriteTimeUtc](#) (string path, DateTime lastWriteTimeUtc)
Sets the date and time, in Coordinated Universal Time (UTC) format, that a directory was last written to.

Static Private Member Functions

- static T [ReturningExclusiveSection< T >](#) (Func< T > action)
A wrapper function for invoking [ConcurrentIO.ExclusiveSection\(Action\)](#) on I/O functions with a return value.

5.3.1 Detailed Description

A lightweight thread-safe directory utility serving as an abstraction over the raw System.IO.Directory class. Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#). Does not handle exceptions.

5.3.2 Member Function Documentation

5.3.2.1 CreateDirectory()

```
DirectoryInfo ProjectGamedev.IO.Directory.CreateDirectory (
    string path) [inline], [static]
```

Creates all directories and subdirectories in the specified path unless they already exist.

Parameters

<i>path</i>	The directory to create.
-------------	--------------------------

Returns

An object that represents the directory at the specified path, or null if the directory does not exist (e.g. if creation failed).

Exceptions

<i>IOException</i>	The directory specified by <i>path</i> is a file, or the network name is not known.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:). You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is null.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>NotSupportedException</i>	<i>path</i> contains a colon character (:) that is not part of a drive label ("C:").

Any and all directories specified in *path* are created, unless they already exist or unless some part of *path* is invalid. If the directory already exists, this method does not create a new directory, and does nothing.

The *path* parameter specifies a directory path, not a file path.

Trailing spaces are removed from the end of the *path* parameter before creating the directory.

You can create a directory on a remote computer, on a share that you have write access to. UNC paths are supported; for example, you can specify the following for *path* : \2009\Archives\December in Visual Basic, and \\2009\Archives\December in C#.

Creating a directory with only the colon character (:) is not supported, and will cause a NotSupportedException to be thrown.

On Unix systems, use a forward slash (/) as path separator.

5.3.2.2 Delete() [1/2]

```
void ProjectGamedev.IO.Directory.Delete (
    string path) [inline], [static]
```

Deletes an empty directory from a specified path.

Parameters

<i>path</i>	The name of the empty directory to remove. This directory must be writable and empty.
-------------	---

Exceptions

<i>IOException</i>	A file with the same name and location specified by <i>path</i> exists, or the directory is the application's current working directory, or the directory specified by <i>path</i> is not empty, or the directory is read-only or contains a read-only file, or the directory is being used by another process.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:). You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	<i>path</i> does not exist or could not be found, or the specified path is invalid (for example, it is on an unmapped drive).

This method behaves identically to `Delete(String, Boolean)` with `false` specified for the second parameter.

The path parameter may specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

Trailing spaces are removed from the end of the *path* parameter before deleting the directory.

This method throws an `IOException` if the directory specified in the path parameter contains files or subdirectories.

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

In some cases, if you have the specified directory open in [File Explorer](#), the `Delete(String)` method may not be able to delete it.

5.3.2.3 Delete() [2/2]

```
void ProjectGamedev.IO.Directory.Delete (
    string path,
    bool recursive) [inline], [static]
```

Deletes the specified directory and, if indicated, any subdirectories and files in the directory.

Parameters

<i>path</i>	The name of the directory to remove.
<i>recursive</i>	<code>true</code> to remove directories, subdirectories, and files in <i>path</i> ; otherwise, <code>false</code> .

Exceptions

<i>IOException</i>	A file with the same name and location specified by <i>path</i> exists, or the directory is the application's current working directory, or the directory specified by <i>path</i> is read-only, or <i>recursive</i> is <code>false</code> and <i>path</i> is not an empty directory, or the directory contains a read-only file, or the directory is being used by another process.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:). You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	<i>path</i> does not exist or could not be found, or the specified path is invalid (for example, it is on an unmapped drive).

The *path* parameter may specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

Trailing spaces are removed from the end of the *path* parameter before deleting the directory.

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

If the *recursive* parameter is `true`, the user must have write permission for the current directory, as well as for all subdirectories.

The behavior of this method differs slightly when deleting a directory that contains a reparse point, such as a symbolic link or a mount point.

If the reparse point is a directory, such as a mount point, it is unmounted and the mount point is deleted.

This method does not recurse through the reparse point.

If the reparse point is a symbolic link to a file, the reparse point is deleted and not the target of the symbolic link.

In some cases, if you have the specified directory open in [File Explorer](#), the `Delete(String)` method may not be able to delete it.

5.3.2.4 EnumerateDirectories() [1/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateDirectories (
    string path) [inline], [static]
```

Returns an enumerable collection of directory full names in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
-------------	--

Returns

An enumerable collection of the full names (including paths) for the directories in the directory specified by *path*.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:). You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The returned directory names are prefixed with the value you provided in the *path* parameter.

For example, if you provide a relative path in the *path* parameter, the returned directory names will contain a relative path.

The EnumerateDirectories(String) and GetDirectories(String) methods differ as follows:

When you use EnumerateDirectories(String), you can start enumerating the collection of names before the whole collection is returned;

when you use GetDirectories(String), you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, EnumerateDirectories(String) can be more efficient.

The returned collection is not cached; each call to the IEnumerable<T>.GetEnumerator on the collection will start a new enumeration.

5.3.2.5 EnumerateDirectories() [2/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateDirectories (
    string path,
    string searchPattern) [inline], [static]
```

Returns an enumerable collection of directory full names that match a search pattern in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of directories in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

Returns

An enumerable collection of the full names (including paths) for the directories in the directory specified by *path* and that match the specified search pattern.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:), or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The returned directory names are prefixed with the value you provided in the *path* parameter.

For example, if you provide a relative path in the *path* parameter, the returned directory names will contain a relative path.

The `EnumerateDirectories(String, String)` and `GetDirectories(String, String)` methods differ as follows:

When you use `EnumerateDirectories(String, String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetDirectories(String, String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateDirectories(String, String)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.6 EnumerateDirectories() [3/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateDirectories (
    string path,
    string searchPattern,
    EnumerationOptions enumerationOptions) [inline], [static]
```

Returns an enumerable collection of the directory full names that match a search pattern in a specified path, and optionally searches subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of directories in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>enumerationOptions</i>	An object that describes the search and enumeration configuration to use.

Returns

An enumerable collection of the full names (including paths) for the directories in the directory specified by *path* and that match the specified search pattern and enumeration options.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>enumerationOptions</i> is not a valid EnumerationOptions value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by Path.DirectorySeparatorChar or Path.AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The returned directory names are prefixed with the value you provided in the *path* parameter.

For example, if you provide a relative path in the *path* parameter, the returned directory names will contain a relative path.

The `EnumerateDirectories(String, String, EnumerationOptions)` and `GetDirectories(String, String, EnumerationOptions)` methods differ as follows:

When you use `EnumerateDirectories(String, String, EnumerationOptions)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetDirectories(String, String, EnumerationOptions)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateDirectories(String, String, EnumerationOptions)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.7 EnumerateDirectories() [4/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateDirectories (
    string path,
    string searchPattern,
    SearchOption searchOption) [inline], [static]
```

Returns an enumerable collection of directory full names that match a search pattern in a specified path, and optionally searches subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of directories in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>searchOption</i>	One of the enumeration values that specifies whether the search operation should include only the current directory or should include all subdirectories. The default value is <code>SearchOption.TopDirectoryOnly</code> .

Returns

An enumerable collection of the full names (including paths) for the directories in the directory specified by *path* and that match the specified search pattern and search option.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>searchOption</i> is not a valid <code>SearchOption</code> value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.

<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*t" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by Path.DirectorySeparatorChar or Path.AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The returned directory names are prefixed with the value you provided in the *path* parameter.

For example, if you provide a relative path in the *path* parameter, the returned directory names will contain a relative path.

The EnumerateDirectories(String, String, SearchOption) and GetDirectories(String, String, SearchOption) methods differ as follows:

When you use EnumerateDirectories(String, String, SearchOption), you can start enumerating the collection of names before the whole collection is returned;

when you use GetDirectories(String, String, SearchOption), you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, EnumerateDirectories(String, String, SearchOption) can be more efficient.

The returned collection is not cached; each call to the IEnumerable<T>.GetEnumerator on the collection will start a new enumeration.

5.3.2.8 EnumerateFiles() [1/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFiles (
    string path) [inline], [static]
```

Returns an enumerable collection of full file names in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
-------------	--

Returns

An enumerable collection of the full names (including paths) for the files in the directory specified by *path* .

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The `EnumerateFiles(String)` and `GetFiles(String)` methods differ as follows:

When you use `EnumerateFiles(String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFiles(String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFiles(String)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.9 EnumerateFiles() [2/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFiles (
    string path,
    string searchPattern) [inline], [static]
```

Returns an enumerable collection of full file names that match a search pattern in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

Returns

An enumerable collection of the full names (including paths) for the files in the directory specified by *path* and that match the specified search pattern.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The `EnumerateFiles(String, String)` and `GetFiles(String, String)` methods differ as follows:

When you use `EnumerateFiles(String, String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFiles(String, String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFiles(String, String)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.10 EnumerateFiles() [3/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFiles (
    string path,
    string searchPattern,
    EnumerationOptions enumerationOptions) [inline], [static]
```

Returns an enumerable collection of full file names that match a search pattern and enumeration options in a specified path, and optionally searches subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>enumerationOptions</i>	An object that describes the search and enumeration configuration to use.

Returns

An enumerable collection of the full names (including paths) for the files in the directory specified by *path* and that match the specified search pattern and enumeration options.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>enumerationOptions</i> is not a valid EnumerationOptions value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by Path.DirectorySeparatorChar or Path.AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The EnumerateFiles(String, String, EnumerationOptions) and GetFiles(String, String, EnumerationOptions) methods differ as follows:

When you use EnumerateFiles(String, String, EnumerationOptions), you can start enumerating the collection of names before the whole collection is returned;

when you use GetFiles(String, String, EnumerationOptions), you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, EnumerateFiles(String, String, EnumerationOptions) can be more efficient.

The returned collection is not cached; each call to the IEnumerable<T>.GetEnumerator on the collection will start a new enumeration.

5.3.2.11 EnumerateFiles() [4/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFiles (
    string path,
    string searchPattern,
    SearchOption searchOption) [inline], [static]
```

Returns an enumerable collection of full file names that match a search pattern in a specified path, and optionally searches subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>searchOption</i>	One of the enumeration values that specifies whether the search operation should include only the current directory or should include all subdirectories. The default value is SearchOption.TopDirectoryOnly.

Returns

An enumerable collection of the full names (including paths) for the files in the directory specified by *path* and that match the specified search pattern and search option.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <i>null</i> , or <i>searchPattern</i> is <i>null</i> .
<i>ArgumentOutOfRangeException</i>	<i>searchOption</i> is not a valid SearchOption value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
------------------------------------	---

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The `EnumerateFiles(String, String, SearchOption)` and `GetFiles(String, String, SearchOption)` methods differ as follows:

When you use `EnumerateDirectories(String, String, SearchOption)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFiles(String, String, SearchOption)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFiles(String, String, SearchOption)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.12 EnumerateFileSystemEntries() [1/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFileSystemEntries (
    string path) [inline], [static]
```

Returns an enumerable collection of file names and directory names in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
-------------	--

Returns

An enumerable collection of file-system entries in the directory specified by *path*.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The `EnumerateFileSystemEntries(String)` and `GetFileSystemEntries(String)` methods differ as follows:

When you use `EnumerateFileSystemEntries(String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFileSystemEntries(String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFileSystemEntries(String)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.13 EnumerateFileSystemEntries() [2/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFileSystemEntries (
    string path,
    string searchPattern) [inline], [static]
```

Returns an enumerable collection of file names and directory names that match a search pattern in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of file-system entries in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

Returns

An enumerable collection of file-system entries in the directory specified by *path* and that match the specified search pattern.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `"*t"` searches for all names in path ending with the letter "t".

The *searchPattern* string `"s*"` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`".."`) or contain two periods (`".."`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The `EnumerateFileSystemEntries(String, String)` and `GetFileSystemEntries(String, String)` methods differ as follows:

When you use `EnumerateFileSystemEntries(String, String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFileSystemEntries(String, String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFileSystemEntries(String, String)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.14 EnumerateFileSystemEntries() [3/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFileSystemEntries (
    string path,
    string searchPattern,
    EnumerationOptions enumerationOptions) [inline], [static]
```

Returns an enumerable collection of file names and directory names that match a search pattern and enumeration options in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files and directories in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>enumerationOptions</i>	An object that describes the search and enumeration configuration to use.

Returns

An enumerable collection of file-system entries in the directory specified by *path* , that match the specified search pattern and the specified enumeration options.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>enumerationOptions</i> is not a valid EnumerationOptions value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by Path.DirectorySeparatorChar or Path.AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The `EnumerateFileSystemEntries(String, String, EnumerationOptions)` and `GetFileSystemEntries(String, String, EnumerationOptions)` methods differ as follows:

When you use `EnumerateFileSystemEntries(String, String, EnumerationOptions)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFileSystemEntries(String, String, EnumerationOptions)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFileSystemEntries(String, String, EnumerationOptions)` can be more efficient.

The returned collection is not cached; each call to the `IEnumerable<T>.GetEnumerator` on the collection will start a new enumeration.

5.3.2.15 EnumerateFileSystemEntries() [4/4]

```
IEnumerable< string > ProjectGamedev.IO.Directory.EnumerateFileSystemEntries (
    string path,
    string searchPattern,
    SearchOption searchOption) [inline], [static]
```

Returns an enumerable collection of file names and directory names that match a search pattern in a specified path, and optionally searches subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of file-system entries in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>searchOption</i>	One of the enumeration values that specifies whether the search operation should include only the current directory or should include all subdirectories. The default value is <code>SearchOption.TopDirectoryOnly</code> .

Returns

An enumerable collection of file-system entries in the directory specified by *path* and that match the specified search pattern and search option.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>searchOption</i> is not a valid <code>SearchOption</code> value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
------------------------------------	---

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string *"*t"* searches for all names in path ending with the letter *"t"*.

The *searchPattern* string *"s*"* searches for all names in path beginning with the letter *"s"*.

searchPattern cannot end in two periods (*".."*) or contain two periods (*".."*) followed by *Path.DirectorySeparatorChar* or *Path.AltDirectorySeparatorChar*, nor can it contain any invalid characters. You can query for invalid characters by using the *Path.GetInvalidPathChars* method.

You can specify relative or absolute path information in the *path* parameter.

Relative path information is interpreted as relative to the current working directory, which you can determine by using the [Directory.GetCurrentDirectory](#) method.

The *EnumerateFileSystemEntries(String, String, SearchOption)* and *GetFileSystemEntries(String, String, SearchOption)* methods differ as follows:

When you use *EnumerateFileSystemEntries(String, String, SearchOption)*, you can start enumerating the collection of names before the whole collection is returned;

when you use *GetFileSystemEntries(String, String, SearchOption)*, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, *EnumerateFileSystemEntries(String, String, SearchOption)* can be more efficient.

The returned collection is not cached; each call to the *IEnumerable<T>.GetEnumerator* on the collection will start a new enumeration.

5.3.2.16 Exists()

```
bool ProjectGamedev.IO.Directory.Exists (
    string path) [inline], [static]
```

Determines whether the given path refers to an existing directory on disk.

Parameters

<i>path</i>	The path to test.
-------------	-------------------

Returns

`true` if `path` refers to an existing directory;
`false` if the directory does not exist or an error occurs when trying to determine if the specified directory exists.

The `path` parameter is permitted to specify relative or absolute path information.
 Relative path information is interpreted as relative to the current working directory.

Trailing spaces are removed from the end of the `path` parameter before checking whether the directory exists.

The case-sensitivity of the `path` parameter corresponds to that of the file system on which the code is running.
 For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

If you do not have at a minimum read-only permission to the directory, the [Exists](#) method will return `false`.

The [Exists](#) method returns `false` if any error occurs while trying to determine if the specified file exists.
 This can occur in situations that raise exceptions such as passing a file name with invalid characters or too many characters,
 a failing or missing disk, or if the caller does not have permission to read the file.

5.3.2.17 GetCreationTime()

```
DateTime ProjectGamedev.IO.Directory.GetCreationTime (
    string path) [inline], [static]
```

Gets the creation date and time of a directory.

Parameters

<i>path</i>	The path of the directory.
-------------	----------------------------

Returns

A structure that is set to the creation date and time for the specified directory. This value is expressed in local time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:). You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

This method is equivalent to [File.GetCreationTime](#).

If the directory described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC), adjusted to local time.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.18 GetCreationTimeUtc()

```
DateTime ProjectGamedev.IO.Directory.GetCreationTimeUtc (
    string path) [inline], [static]
```

Gets the creation date and time, in Coordinated Universal Time (UTC) format, of a directory.

Parameters

<i>path</i>	The path of the directory.
-------------	----------------------------

Returns

A structure that is set to the creation date and time for the specified directory. This value is expressed in UTC time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:). You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

If the directory described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

Use this method to get the creation time for a directory based on Coordinated Universal Time (UTC).

5.3.2.19 GetCurrentDirectory()

```
string ProjectGamedev.IO.Directory.GetCurrentDirectory () [inline], [static]
```

Gets the current working directory of the application.

Returns

A string that contains the absolute path of the current working directory, and does not end with a backslash ().

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>NotSupportedException</i>	The operating system is Windows CE, which does not have current directory functionality.

The current directory is distinct from the original directory, which is the one from which the process was started.

5.3.2.20 GetDirectories() [1/4]

```
string[] ProjectGamedev.IO.Directory.GetDirectories (
    string path) [inline], [static]
```

Returns the names of subdirectories (including their paths) in the specified directory.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
-------------	--

Returns

An array of the full names (including paths) of subdirectories in the specified path, or an empty array if no directories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>path</i> is prefixed with, or contains, only a colon character (:). You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or file or directory is corrupted and unreadable (example: invalid first allocation unit of a FAT32 partition).
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

This method is identical to `GetDirectories(String, String)` with the asterisk (*) specified as the search pattern, so it returns all subdirectories.

If you need to search subdirectories, use the `GetDirectories(String, String, SearchOption)` method, which enables you to specify a search of subdirectories with the `searchOption` parameter.

The `EnumerateDirectories(String)` and `GetDirectories(String)` methods differ as follows:

When you use `EnumerateDirectories(String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetDirectories(String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateDirectories(String)` can be more efficient.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The names returned by this method are prefixed with the directory information provided in *path*.

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running.

For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.21 GetDirectories() [2/4]

```
string[] ProjectGamedev.IO.Directory.GetDirectories (
    string path,
    string searchPattern) [inline], [static]
```

Returns the names of subdirectories (including their paths) that match the specified search pattern in the specified directory.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of directories in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

Returns

An array of the full names (including paths) of the subdirectories that match the search pattern in the specified directory, or an empty array if no directories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
--------------------------	---

Exceptions

<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or file or directory is corrupted and unreadable (example: invalid first allocation unit of a FAT32 partition).
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

This method returns all subdirectories directly under the specified directory that match the specified search pattern. If the specified directory has no subdirectories, or no subdirectories match the *searchPattern* parameter, this method returns an empty array.

Only the top directory is searched.

If you want to search the subdirectories as well, use the `GetDirectories(String, String, SearchOption)` method and specify `SearchOption.AllDirectories` in the *searchOption* parameter.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `"*t"` searches for all names in path ending with the letter "t".

The *searchPattern* string `"s*"` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`".."`) or contain two periods (`".."`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The `EnumerateDirectories(String, String)` and `GetDirectories(String, String)` methods differ as follows:

When you use `EnumerateDirectories(String, String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetDirectories(String, String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateDirectories(String, String)` can be more efficient.

5.3.2.22 GetDirectories() [3/4]

```
string[] ProjectGamedev.IO.Directory.GetDirectories (
    string path,
    string searchPattern,
    EnumerationOptions enumerationOptions) [inline], [static]
```

Returns the names of subdirectories (including their paths) that match the specified search pattern and enumeration options in the specified directory.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of directories in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>enumerationOptions</i>	An object that describes the search and enumeration configuration to use.

Returns

An array of the full names (including paths) of the subdirectories that match the search pattern and enumeration options in the specified directory, or an empty array if no directories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or file or directory is corrupted and unreadable (example: invalid first allocation unit of a FAT32 partition).
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

This method returns all subdirectories directly under the specified directory that match the specified search pattern. If the specified directory has no subdirectories, or no subdirectories match the *searchPattern* parameter, this method returns an empty array.

Only the top directory is searched.

If you want to search the subdirectories as well, use the GetDirectories(String, String, SearchOption) method and specify SearchOption.AllDirectories in the searchOption parameter.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by Path.DirectorySeparatorChar or Path.AltDirectorySeparatorChar, nor can it contain any invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.

The *path* parameter can specify relative or absolute path information.
 Relative path information is interpreted as relative to the current working directory.
 To obtain the current working directory, see [GetCurrentDirectory](#).

The `EnumerateDirectories(String, String, EnumerationOptions)` and `GetDirectories(String, String, EnumerationOptions)` methods differ as follows:

When you use `EnumerateDirectories(String, String, EnumerationOptions)`, you can start enumerating the collection of names before the whole collection is returned;
 when you use `GetDirectories(String, String, EnumerationOptions)`, you must wait for the whole array of names to be returned before you can access the array.
 Therefore, when you are working with many files and directories, `EnumerateDirectories(String, String, EnumerationOptions)` can be more efficient.

5.3.2.23 GetDirectories() [4/4]

```
string[] ProjectGamedev.IO.Directory.GetDirectories (
    string path,
    string searchPattern,
    SearchOption searchOption) [inline], [static]
```

Returns the names of the subdirectories (including their paths) that match the specified search pattern in the specified directory, and optionally searches subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of directories in <i>path</i> . This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>searchOption</i>	One of the enumeration values that specifies whether the search operation should include all subdirectories or only the current directory.

Returns

An array of the full names (including paths) of the subdirectories that match the specified criteria, or an empty array if no directories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> does not contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> , or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>searchOption</i> is not a valid <code>SearchOption</code> value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or file or directory is corrupted and unreadable (example: invalid first allocation unit of a FAT32 partition).
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
------------------------------------	---

This method returns all subdirectories directly under the specified directory that match the specified search pattern. If the specified directory has no subdirectories, or no subdirectories match the *searchPattern* parameter, this method returns an empty array.

Only the top directory is searched.

If you want to search the subdirectories as well, use the `GetDirectories(String, String, SearchOption)` method and specify `SearchOption.AllDirectories` in the *searchOption* parameter.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string "*" searches for all names in path ending with the letter "t".

The *searchPattern* string "s*" searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods ("..") or contain two periods ("..") followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The `EnumerateDirectories(String, String, SearchOption)` and `GetDirectories(String, String, SearchOption)` methods differ as follows:

When you use `EnumerateDirectories(String, String, SearchOption)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetDirectories(String, String, SearchOption)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateDirectories(String, String, SearchOption)` can be more efficient.

5.3.2.24 GetDirectoryRoot()

```
string ProjectGamedev.IO.Directory.GetDirectoryRoot (
    string path) [inline], [static]
```

Returns the volume information, root information, or both for the specified path.

Parameters

<i>path</i>	The path of a file or directory.
-------------	----------------------------------

Returns

A string that contains the volume information, root information, or both for the specified path.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

This method obtains the fully qualified path name of *path*, as returned by `Path.GetFullPath(String)`, and returns root directory information.

The specified path is not required to exist.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.25 GetFiles() [1/4]

```
string[] ProjectGamedev.IO.Directory.GetFiles (
    string path) [inline], [static]
```

Returns the names of files (including their paths) in the specified directory.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
-------------	--

Returns

An array of the full names (including paths) for the files in the specified directory, or an empty array if no files are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or a network error has occurred.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The EnumerateFiles(String) and GetFiles(String) methods differ as follows:

When you use EnumerateFiles(String), you can start enumerating the collection of names before the whole collection is returned;

when you use GetFiles(String), you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, EnumerateFiles(String) can be more efficient.

The returned file names are appended to the supplied *path* parameter.

This method is identical to GetFiles(String, String) with the asterisk (*) specified as the search pattern.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The order of the returned file names is not guaranteed; use the Array.Sort(Array) method if a specific sort order is required.

The case-sensitivity of the path parameter corresponds to that of the file system on which the code is running.

For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.26 GetFiles() [2/4]

```
string[] ProjectGamedev.IO.Directory.GetFiles (
    string path,
    string searchPattern) [inline], [static]
```

Returns the names of files (including their paths) that match the specified search pattern in the specified directory.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files in path. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

Returns

An array of the full names (including paths) for the files in the specified directory that match the specified search pattern, or an empty array if no files are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> doesn't contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> or <i>searchPattern</i> is null.

<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or a network error has occurred.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The order of the returned file names is not guaranteed; use the `Array.Sort(Array)` method if a specific sort order is required.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `"*t"` searches for all names in path ending with the letter "t".

The *searchPattern* string `"s*"` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`".."`) or contain two periods (`".."`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The `EnumerateFiles(String, String)` and `GetFiles(String, String)` methods differ as follows:

When you use `EnumerateFiles(String, String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFiles(String, String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFiles(String, String)` can be more efficient.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the path parameter corresponds to that of the file system on which the code is running.

For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.27 GetFiles() [3/4]

```
string[] ProjectGamedev.IO.Directory.GetFiles (
    string path,
    string searchPattern,
    EnumerationOptions enumerationOptions) [inline], [static]
```

Returns the names of files (including their paths) that match the specified search pattern and enumeration options in the specified directory.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files in path. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>enumerationOptions</i>	An object that describes the search and enumeration configuration to use.

Returns

An array of the full names (including paths) for the files in the specified directory that match the specified search pattern and enumeration options, or an empty array if no files are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> doesn't contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or a network error has occurred.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The order of the returned file names is not guaranteed; use the `Array.Sort(Array)` method if a specific sort order is required.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `*t` searches for all names in path ending with the letter "t".

The *searchPattern* string `s*` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`..`) or contain two periods (`..`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The `EnumerateFiles(String, String, EnumerationOptions)` and `GetFiles(String, String, EnumerationOptions)` methods differ as follows:

When you use `EnumerateFiles(String, String, EnumerationOptions)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFiles(String, String, EnumerationOptions)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFiles(String, String, EnumerationOptions)` can be more efficient.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the path parameter corresponds to that of the file system on which the code is running.

For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.28 GetFiles() [4/4]

```
string[] ProjectGamedev.IO.Directory.GetFiles (
    string path,
    string searchPattern,
    SearchOption searchOption) [inline], [static]
```

Returns the names of files (including their paths) that match the specified search pattern in the specified directory, using a value to determine whether to search subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files in path. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>searchOption</i>	One of the enumeration values that specifies whether the search operation should include all subdirectories or only the current directory.

Returns

An array of the full names (including paths) for the files in the specified directory that match the specified search pattern and option, or an empty array if no files are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> doesn't contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>searchOption</i> is not a valid <code>SearchOption</code> value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name, or a network error has occurred.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The order of the returned file names is not guaranteed; use the `Array.Sort(Array)` method if a specific sort order is required.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `"*t"` searches for all names in path ending with the letter "t".

The *searchPattern* string `"s*"` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`".."`) or contain two periods (`".."`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The `EnumerateFiles(String, String, SearchOption)` and `GetFiles(String, String, SearchOption)` methods differ as follows:

When you use `EnumerateFiles(String, String, SearchOption)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFiles(String, String, SearchOption)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFiles(String, String, SearchOption)` can be more efficient.

The file names include the full path.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the path parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.29 GetFileSystemEntries() [1/4]

```
string[] ProjectGamedev.IO.Directory.GetFileSystemEntries (
    string path) [inline], [static]
```

Returns the names of all files and subdirectories in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
-------------	--

Returns

An array of the names of files and subdirectories in the specified directory, or an empty array if no files or subdirectories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The order of the returned file names is not guaranteed; use the `Array.Sort(Array)` method if a specific sort order is required.

The `EnumerateFileSystemEntries(String)` and `GetFileSystemEntries(String)` methods differ as follows: When you use `EnumerateFileSystemEntries(String)`, you can start enumerating the collection of names before the whole collection is returned; when you use `GetFileSystemEntries(String)`, you must wait for the whole array of names to be returned before you can access the array. Therefore, when you are working with many files and directories, `EnumerateFileSystemEntries(String)` can be more efficient.

This method is identical to `GetFileSystemEntries(String, String)` with the asterisk (*) specified as the search pattern.

The *path* parameter can specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.30 GetFileSystemEntries() [2/4]

```
string[] ProjectGamedev.IO.Directory.GetFileSystemEntries (
    string path,
    string searchPattern) [inline], [static]
```

Returns an array of file names and directory names that match a search pattern in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files and directories in path. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.

Returns

An array of file names and directory names that match the specified search criteria, or an empty array if no files or directories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> doesn't contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The order of the returned file names is not guaranteed; use the `Array.Sort(Array)` method if a specific sort order is required.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `"*t"` searches for all names in path ending with the letter "t".

The *searchPattern* string `"s*"` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`".."`) or contain two periods (`".."`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The `EnumerateFileSystemEntries(String, String)` and `GetFileSystemEntries(String, String)` methods differ as follows:

When you use `EnumerateFileSystemEntries(String, String)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFileSystemEntries(String, String)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFileSystemEntries(String, String)` can be more efficient.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the path parameter corresponds to that of the file system on which the code is running.

For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.31 GetFileSystemEntries() [3/4]

```
string[] ProjectGamedev.IO.Directory.GetFileSystemEntries (
    string path,
    string searchPattern,
    EnumerationOptions enumerationOptions) [inline], [static]
```

Returns an array of file names and directory names that match a search pattern and enumeration options in a specified path.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files and directories in path. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>enumerationOptions</i>	An object that describes the search and enumeration configuration to use.

Returns

An array of file names and directory names that match the specified search pattern and enumeration options, or an empty array if no files or directories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> doesn't contain a valid pattern. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> or <i>searchPattern</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The order of the returned file names is not guaranteed; use the `Array.Sort(Array)` method if a specific sort order is required.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `"*t"` searches for all names in path ending with the letter "t".

The *searchPattern* string `"s*"` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`".."`) or contain two periods (`".."`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The `EnumerateFileSystemEntries(String, String, EnumerationOptions)` and `GetFileSystemEntries(String, String, EnumerationOptions)` methods differ as follows:

When you use `EnumerateFileSystemEntries(String, String, EnumerationOptions)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFileSystemEntries(String, String, EnumerationOptions)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFileSystemEntries(String, String, EnumerationOptions)` can be more efficient.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the path parameter corresponds to that of the file system on which the code is running.

For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.32 GetFileSystemEntries() [4/4]

```
string[] ProjectGamedev.IO.Directory.GetFileSystemEntries (
    string path,
    string searchPattern,
    SearchOption searchOption) [inline], [static]
```

Returns an array of all the file names and directory names that match a search pattern in a specified path, and optionally searches subdirectories.

Parameters

<i>path</i>	The relative or absolute path to the directory to search. This string is not case-sensitive.
<i>searchPattern</i>	The search string to match against the names of files and directories in path. This parameter can contain a combination of valid literal path and wildcard (* and ?) characters, but it doesn't support regular expressions.
<i>searchOption</i>	One of the enumeration values that specifies whether the search operation should include only the current directory or should include all subdirectories. The default value is <code>SearchOption.TopDirectoryOnly</code> .

Returns

An array of file the file names and directory names that match the specified search criteria, or an empty array if no files or directories are found.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>searchPattern</i> doesn't contain a valid pattern. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> or <i>searchPattern</i> is <code>null</code> .
<i>ArgumentOutOfRangeException</i>	<i>searchOption</i> is not a valid <code>SearchOption</code> value.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid, such as referring to an unmapped drive.
<i>IOException</i>	<i>path</i> is a file name.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The order of the returned file names is not guaranteed; use the `Array.Sort(Array)` method if a specific sort order is required.

searchPattern can be a combination of literal and wildcard characters, but it doesn't support regular expressions. The following wildcard specifiers are permitted in *searchPattern* :

- * (asterisk) - Zero or more characters in that position.
- ? (question mark) - Exactly one character in that position.

Characters other than the wildcard are literal characters.

For example, the *searchPattern* string `"*t"` searches for all names in path ending with the letter "t".

The *searchPattern* string `"s*"` searches for all names in path beginning with the letter "s".

searchPattern cannot end in two periods (`".."`) or contain two periods (`".."`) followed by `Path.DirectorySeparatorChar` or `Path.AltDirectorySeparatorChar`, nor can it contain any invalid characters. You can query for invalid characters by using the `Path.GetInvalidPathChars` method.

The `EnumerateFileSystemEntries(String, String, SearchOption)` and `GetFileSystemEntries(String, String, SearchOption)` methods differ as follows:

When you use `EnumerateFileSystemEntries(String, String, SearchOption)`, you can start enumerating the collection of names before the whole collection is returned;

when you use `GetFileSystemEntries(String, String, SearchOption)`, you must wait for the whole array of names to be returned before you can access the array.

Therefore, when you are working with many files and directories, `EnumerateFileSystemEntries(String, String, SearchOption)` can be more efficient.

The file names include the full path.

The *path* parameter can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [GetCurrentDirectory](#).

5.3.2.33 GetLastAccessTime()

```
DateTime ProjectGamedev.IO.Directory.GetLastAccessTime (
    string path) [inline], [static]
```

Returns the date and time the specified file or directory was last accessed.

Parameters

<i>path</i>	The file or directory for which to obtain access date and time information.
-------------	---

Returns

A structure that is set to the date and time the specified file or directory was last accessed. This value is expressed in local time.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>NotSupportedException</i>	The path parameter is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

This method is equivalent to [File.GetLastAccessTime](#).

If the directory described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC), adjusted to local time.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.34 GetLastAccessTimeUtc()

```
DateTime ProjectGamedev.IO.Directory.GetLastAccessTimeUtc (
    string path) [inline], [static]
```

Returns the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last accessed.

Parameters

<i>path</i>	The file or directory for which to obtain access date and time information.
-------------	---

Returns

A structure that is set to the date and time the specified file or directory was last accessed. This value is expressed in UTC time.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>NotSupportedException</i>	The path parameter is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

If the directory described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.35 GetLastWriteTime()

```
DateTime ProjectGamedev.IO.Directory.GetLastWriteTime (
    string path) [inline], [static]
```

Returns the date and time the specified file or directory was last written to.

Parameters

<i>path</i>	The file or directory for which to obtain modification date and time information.
-------------	---

Returns

A structure that is set to the date and time the specified file or directory was last written to. This value is expressed in local time.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

If the directory described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC), adjusted to local time.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.36 GetLastWriteTimeUtc()

```
DateTime ProjectGamedev.IO.Directory.GetLastWriteTimeUtc (
    string path) [inline], [static]
```

Returns the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last written to.

Parameters

<i>path</i>	The file or directory for which to obtain modification date and time information.
-------------	---

Returns

A structure that is set to the date and time the specified file or directory was last written to. This value is expressed in UTC time.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

If the directory described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.37 GetLogicalDrives()

```
string[] ProjectGamedev.IO.Directory.GetLogicalDrives () [inline], [static]
```

Retrieves the names of the logical drives on this computer in the form "<drive letter>:\".

Returns

The logical drives on this computer.

Exceptions

<i>IOException</i>	An I/O error occurred (for example, a disk error).
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

GetLogicalDrives returns all the accessible drives on a particular machine, including the floppy drive and any optical drives.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.38 GetParent()

```
DirectoryInfo ProjectGamedev.IO.Directory.GetParent (
    string path) [inline], [static]
```

Retrieves the parent directory of the specified path, including both absolute and relative paths.

Parameters

<i>path</i>	The path for which to retrieve the parent directory.
-------------	--

Returns

The parent directory, or null if path is the root directory, including the root of a UNC server or share name.

Exceptions

<i>IOException</i>	The directory specified by <i>path</i> is read-only.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is null.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>DirectoryNotFoundException</i>	The specified path was not found.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	.NET Framework only: The caller does not have the required permissions.

The *path* parameter can specify relative or absolute path information.
 Relative path information is interpreted as relative to the current working directory.
 To obtain the current working directory, see [GetCurrentDirectory](#).

Trailing spaces are removed from the end of the path parameter before getting the directory.

The string returned by this method consists of all characters in the path up to, but not including, the last Path.DirectorySeparatorChar or Path.AltDirectorySeparatorChar.
 For example, passing the path "C:/Directory/SubDirectory/test.txt" to [GetParent](#) returns "C:/Directory/SubDirectory".
 Passing "C:/Directory/SubDirectory" returns "C:/Directory".
 However, passing "C:/Directory/SubDirectory/" returns "C:/Directory/SubDirectory", because the ending directory separator is after "SubDirectory".

The case-sensitivity of the path parameter corresponds to that of the file system on which the code is running.
 For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.39 Move()

```
void ProjectGamedev.IO.Directory.Move (
    string sourceDirName,
    string destDirName) [inline], [static]
```

Moves a file or a directory and its contents to a new location.

Parameters

<i>sourceDirName</i>	The path of the file or directory to move.
<i>destDirName</i>	The path to the new location for <i>sourceDirName</i> or its contents. If <i>sourceDirName</i> is a file, then <i>destDirName</i> must also be a file name.

Exceptions

<i>IOException</i>	An attempt was made to move a directory to a different volume, or <i>destDirName</i> already exists (see the note in the Remarks section), or the <i>sourceDirName</i> and <i>destDirName</i> parameters refer to the same file or directory, or the directory or a file within it is being used by another process.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>sourceDirName</i> or <i>destDirName</i> is a zero-length string, contains only white space, contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>sourceDirName</i> or <i>destDirName</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>DirectoryNotFoundException</i>	The path specified by <i>sourceDirName</i> is invalid (for example, it is on an unmapped drive).

This method creates a new directory with the name specified by *destDirName* and moves the contents of *sourceDirName*, including files and directories, to the newly created destination directory. It then deletes the *sourceDirName* directory.

If you try to move a directory to a directory that already exists, an `IOException` will occur.

The *sourceDirName* and *destDirName* arguments are permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

Trailing spaces are removed from the end of the path parameters before moving the directory.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.3.2.40 ReturningExclusiveSection< T >()

```
T ProjectGamedev.IO.Directory.ReturningExclusiveSection< T > (
    Func< T > action) [inline], [static], [private]
```

A wrapper function for invoking [ConcurrentIO.ExclusiveSection\(Action\)](#) on I/O functions with a return value.

Parameters

<i>action</i>	The callable action to execute in a thread-safe manner.
---------------	---

Template Parameters

<i>T</i>	The type of the return value of <i>action</i> .
----------	---

Returns

The return value of *action*.

5.3.2.41 SetCreationTime()

```
void ProjectGamedev.IO.Directory.SetCreationTime (
    string path,
    DateTime creationTime) [inline], [static]
```

Sets the creation date and time for the specified file or directory.

Parameters

<i>path</i>	The file or directory for which to set the creation date and time information.
<i>creationTime</i>	The date and time the file or directory was last written to. This value is expressed in local time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentOutOfRangeException</i>	<i>creationTime</i> specifies a value outside the range of dates or times permitted for this operation.
<i>PlatformNotSupportedException</i>	The current operating system is not Windows NT or later.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.42 SetCreationTimeUtc()

```
void ProjectGamedev.IO.Directory.SetCreationTimeUtc (
    string path,
    DateTime creationTimeUtc) [inline], [static]
```

Sets the creation date and time, in Coordinated Universal Time (UTC) format, for the specified file or directory.

Parameters

<i>path</i>	The file or directory for which to set the creation date and time information.
<i>creationTimeUtc</i>	The date and time the file or directory was last written to. This value is expressed in UTC time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentOutOfRangeException</i>	<i>creationTimeUtc</i> specifies a value outside the range of dates or times permitted for this operation.
<i>PlatformNotSupportedException</i>	The current operating system is not Windows NT or later.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.43 SetCurrentDirectory()

```
void ProjectGamedev.IO.Directory.SetCurrentDirectory (
    string path) [inline], [static]
```

Sets the application's current working directory to the specified directory.

Parameters

<i>path</i>	The path to which the current working directory is set.
-------------	---

Exceptions

<i>IOException</i>	An I/O error occurred.
<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>SecurityException</i>	The caller does not have the required permission to access unmanaged code.
<i>DirectoryNotFoundException</i>	The specified directory was not found.

When the application terminates, the working directory is restored to its original location (the directory where the process was started).

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

Trailing spaces are removed from the end of the *path* parameter before setting the directory.

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

If you are setting the directory to a drive with removable media (for example, "E:" for a USB flash drive), you can determine whether the drive is ready by using the `DriveInfo.IsReady` property.

5.3.2.44 SetLastAccessTime()

```
void ProjectGamedev.IO.Directory.SetLastAccessTime (
    string path,
    DateTime lastAccessTime) [inline], [static]
```

Sets the date and time the specified file or directory was last accessed.

Parameters

<i>path</i>	The file or directory for which to set the access date and time information.
<i>lastAccessTime</i>	An object that contains the value to set for the access date and time of <i>path</i> . This value is expressed in local time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentOutOfRangeException</i>	<i>lastAccessTime</i> specifies a value outside the range of dates or times permitted for this operation.
<i>PlatformNotSupportedException</i>	The current operating system is not Windows NT or later.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.45 SetLastAccessTimeUtc()

```
void ProjectGamedev.IO.Directory.SetLastAccessTimeUtc (
    string path,
    DateTime lastAccessTimeUtc) [inline], [static]
```

Sets the date and time, in Coordinated Universal Time (UTC) format, that the specified file or directory was last accessed.

Parameters

<i>path</i>	The file or directory for which to set the access date and time information.
<i>lastAccessTimeUtc</i>	An object that contains the value to set for the access date and time of <i>path</i> . This value is expressed in UTC time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the Path.GetInvalidPathChars method.

Exceptions

<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentOutOfRangeException</i>	<i>lastAccessTimeUtc</i> specifies a value outside the range of dates or times permitted for this operation.
<i>PlatformNotSupportedException</i>	The current operating system is not Windows NT or later.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.46 SetLastWriteTime()

```
void ProjectGamedev.IO.Directory.SetLastWriteTime (
    string path,
    DateTime lastWriteTime) [inline], [static]
```

Sets the date and time a directory was last written to.

Parameters

<i>path</i>	The path of the directory.
<i>lastWriteTime</i>	The date and time the directory was last written to. This value is expressed in local time.

Exceptions

<i>FileNotFoundException</i>	<i>path</i> was not found (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>DirectoryNotFoundException</i>	<i>path</i> was not found (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentOutOfRangeException</i>	<i>lastWriteTime</i> specifies a value outside the range of dates or times permitted for this operation.
<i>PlatformNotSupportedException</i>	The current operating system is not Windows NT or later.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

5.3.2.47 SetLastWriteTimeUtc()

```
void ProjectGamedev.IO.Directory.SetLastWriteTimeUtc (
    string path,
    DateTime lastWriteTimeUtc) [inline], [static]
```

Sets the date and time, in Coordinated Universal Time (UTC) format, that a directory was last written to.

Parameters

<i>path</i>	The path of the directory.
<i>lastWriteTimeUtc</i>	The date and time the directory was last written to. This value is expressed in UTC time.

Exceptions

<i>FileNotFoundException</i>	<i>path</i> was not found (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>DirectoryNotFoundException</i>	<i>path</i> was not found (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters with the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentOutOfRangeException</i>	<i>lastWriteTimeUtc</i> specifies a value outside the range of dates or times permitted for this operation.
<i>PlatformNotSupportedException</i>	The current operating system is not Windows NT or later.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [GetCurrentDirectory](#).

The case-sensitivity of the *path* parameter corresponds to that of the file system on which the code is running. For example, it's case-insensitive on NTFS (the default Windows file system) and case-sensitive on Linux file systems.

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/[Directory.cs](#)

5.4 ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest Class Reference

Test class containing test cases for the [Directory](#) class.

Public Member Functions

- void [BeforeAll](#) ()
- void [TestDirectoryMethodsNoTimeoutsNoExceptions](#) ()
Test ID: IO-7.

Private Member Functions

- void [CallAllDirectoryMethods](#) ()

Private Attributes

- string [testfilepath1](#)
- string [testfilepath2](#)
- string [testdirectorypath1](#)
- string [testdirectorypath2](#)

5.4.1 Detailed Description

Test class containing test cases for the [Directory](#) class.

5.4.2 Member Function Documentation

5.4.2.1 BeforeAll()

```
void ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest.BeforeAll () [inline]
```

5.4.2.2 CallAllDirectoryMethods()

```
void ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest.CallAllDirectoryMethods () [inline],  
[private]
```

5.4.2.3 TestDirectoryMethodsNoTimeoutsNoExceptions()

```
void ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest.TestDirectoryMethodsNoTimeoutsNo↵  
Exceptions () [inline]
```

Test ID: IO-7.

All supported methods of the `System.IO.Directory` class are called together to assert no deadlocks, timeouts, or other [IO](#) exceptions occur.

5.4.3 Member Data Documentation

5.4.3.1 testdirectorypath1

```
string ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest.testdirectorypath1 [private]
```

5.4.3.2 testdirectorypath2

```
string ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest.testdirectorypath2 [private]
```

5.4.3.3 testfilepath1

```
string ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest.testfilepath1 [private]
```

5.4.3.4 testfilepath2

```
string ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest.testfilepath2 [private]
```

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/[DirectoryTest.cs](#)

5.5 ProjectGamedev.IO.File Class Reference

A lightweight thread-safe file utility serving as an abstraction over the raw System.IO.File class. Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#). Does not handle exceptions.

Static Public Member Functions

- static void [AppendAllLines](#) (string path, IEnumerable< string > contents)
*Appends lines to a file, and then closes the file.
If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file.*
- static void [AppendAllLines](#) (string path, IEnumerable< string > contents, Encoding encoding)
*Appends lines to a file by using a specified encoding, and then closes the file.
If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file.*
- static void [AppendAllText](#) (string path, string contents)
*Opens a file, appends the specified string to the file, and then closes the file.
If the file does not exist, this method creates a file, writes the specified string to the file, then closes the file.*
- static void [AppendAllText](#) (string path, string contents, Encoding encoding)
Appends the specified string to the file using the specified encoding, creating the file if it does not already exist.
- static void [Copy](#) (string sourceFileName, string destFileName, bool overwrite)
Copies an existing file to a new file. Overwriting a file of the same name is allowed.
- static void [Copy](#) (string sourceFileName, string destFileName)

- Copies an existing file to a new file. Overwriting a file of the same name is not allowed.*

 - static void [Create](#) (string path)

Creates, or truncates and overwrites, a file in the specified path.
 - static void [Create](#) (string path, int bufferSize)

Creates, or truncates and overwrites, a file in the specified path, specifying a buffer size.
 - static void [Create](#) (string path, int bufferSize, FileOptions options)

Creates or overwrites a file in the specified path, specifying a buffer size and options that describe how to create or overwrite the file.
 - static void [Delete](#) (string path)

Deletes the specified file.
 - static bool [Exists](#) (string path)

Determines whether the specified file exists.
 - static FileAttributes [GetAttributes](#) (string path)

Gets the FileAttributes of the file on the path.
 - static DateTime [GetCreationTime](#) (string path)

Returns the creation date and time of the specified file or directory.
 - static DateTime [GetCreationTimeUtc](#) (string path)

Returns the creation date and time, in Coordinated Universal Time (UTC), of the specified file or directory.
 - static DateTime [GetLastAccessTime](#) (string path)

Returns the date and time the specified file or directory was last accessed.
 - static DateTime [GetLastAccessTimeUtc](#) (string path)

Returns the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last accessed.
 - static DateTime [GetLastWriteTime](#) (string path)

Returns the date and time the specified file or directory was last written to.
 - static DateTime [GetLastWriteTimeUtc](#) (string path)

Returns the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last written to.
 - static void [Move](#) (string sourceFileName, string destFileName)

Moves a specified file to a new location, providing the option to specify a new file name.
 - static byte[] [ReadAllBytes](#) (string path)

Opens a binary file, reads the contents of the file into a byte array, and then closes the file.
 - static string[] [ReadAllLines](#) (string path)

Opens a text file, reads all lines of the file, and then closes the file.
 - static string[] [ReadAllLines](#) (string path, Encoding encoding)

Opens a file, reads all lines of the file with the specified encoding, and then closes the file.
 - static string [ReadAllText](#) (string path)

Opens a text file, reads all the text in the file, and then closes the file.
 - static string [ReadAllText](#) (string path, Encoding encoding)

Opens a file, reads all text in the file with the specified encoding, and then closes the file.
 - static void [Replace](#) (string sourceFileName, string destinationFileName, string? destinationBackupFileName)

Replaces the contents of a specified file with the contents of another file, deleting the original file and creating a backup of the replaced file.
 - static void [Replace](#) (string sourceFileName, string destinationFileName, string? destinationBackupFileName, bool ignoreMetadataErrors)

Replaces the contents of a specified file with the contents of another file, deleting the original file and creating a backup of the replaced file, and optionally ignores merge errors.
 - static void [SetAttributes](#) (string path, FileAttributes fileAttributes)

Sets the specified FileAttributes of the file on the specified path.
 - static void [SetCreationTime](#) (string path, DateTime creationTime)

Sets the date and time the file was created.
 - static void [SetCreationTimeUtc](#) (string path, DateTime creationTimeUtc)

Sets the date and time, in Coordinated Universal Time (UTC), that the file was created.

- static void [SetLastAccessTime](#) (string path, DateTime lastAccessTime)
Sets the date and time the specified file was last accessed.
- static void [SetLastAccessTimeUtc](#) (string path, DateTime lastAccessTimeUtc)
Sets the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last accessed.
- static void [SetLastWriteTime](#) (string path, DateTime lastWriteTime)
Sets the date and time that the specified file was last written to.
- static void [SetLastWriteTimeUtc](#) (string path, DateTime lastWriteTimeUtc)
Sets the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last written to.
- static void [WriteAllBytes](#) (string path, byte[] bytes)
*Creates a new file, writes the specified byte array to the file, and then closes the file.
If the target file already exists, it is truncated and overwritten.*
- static void [WriteAllLines](#) (string path, string[] contents, Encoding encoding)
Creates a new file, writes the specified string array to the file by using the specified encoding, and then closes the file.
- static void [WriteAllLines](#) (string path, IEnumerable< string > contents, Encoding encoding)
Creates a new file by using the specified encoding, writes a collection of strings to the file, and then closes the file.
- static void [WriteAllLines](#) (string path, IEnumerable< string > contents)
Creates a new file, writes a collection of strings to the file, and then closes the file.
- static void [WriteAllLines](#) (string path, string[] contents)
Creates a new file, write the specified string array to the file, and then closes the file.
- static void [WriteAllText](#) (string path, string contents)
*Creates a new file, writes the specified string to the file, and then closes the file.
If the target file already exists, it is truncated and overwritten.*
- static void [WriteAllText](#) (string path, string contents, Encoding encoding)
*Creates a new file, writes the specified string to the file using the specified encoding, and then closes the file.
If the target file already exists, it is truncated and overwritten.*

Static Private Member Functions

- static T [ReturningCriticalSection< T >](#) (string path, Func< T > action)
A wrapper function for invoking ConcurrentIO.CriticalSection(String, Action) on I/O functions with a return value.

5.5.1 Detailed Description

A lightweight thread-safe file utility serving as an abstraction over the raw System.IO.File class. Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#). Does not handle exceptions.

5.5.2 Member Function Documentation

5.5.2.1 AppendAllLines() [1/2]

```
void ProjectGamedev.IO.File.AppendAllLines (
    string path,
    IEnumerable< string > contents) [inline], [static]
```

Appends lines to a file, and then closes the file.

If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file.

Parameters

<i>path</i>	The file to append the lines to. The file is created if it doesn't already exist.
<i>contents</i>	The lines to append to the file.

Exceptions

<i>ArgumentException</i>	<i>path</i> is a zero-length string, contains only white space, or contains one more invalid characters defined by the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	Either <i>path</i> or <i>contents</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>FileNotFoundException</i>	The file specified by <i>path</i> was not found.
<i>IOException</i>	An I/O error occurred while opening the file.
<i>PathTooLongException</i>	<i>path</i> exceeds the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have permission to write to the file.
<i>UnauthorizedAccessException</i>	<i>path</i> specifies a file that is read-only, or this operation is not supported on the current platform, or <i>path</i> is a directory.

The method creates the file if it doesn't exist, but it doesn't create new directories. Therefore, the value of *path* must contain existing directories.

5.5.2.2 AppendAllLines() [2/2]

```
void ProjectGamedev.IO.File.AppendAllLines (
    string path,
    IEnumerable< string > contents,
    Encoding encoding) [inline], [static]
```

Appends lines to a file by using a specified encoding, and then closes the file. If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file.

Parameters

<i>path</i>	The file to append the lines to. The file is created if it doesn't already exist.
<i>contents</i>	The lines to append to the file.
<i>encoding</i>	The character encoding to use.

Exceptions

<i>ArgumentException</i>	<i>path</i> is a zero-length string, contains only white space, or contains one more invalid characters defined by the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	Either <i>path</i> , <i>contents</i> , or <i>encoding</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>FileNotFoundException</i>	The file specified by <i>path</i> was not found.
<i>IOException</i>	An I/O error occurred while opening the file.
<i>PathTooLongException</i>	<i>path</i> exceeds the system-defined maximum length.

Exceptions

<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have permission to write to the file.
<i>UnauthorizedAccessException</i>	<i>path</i> specifies a file that is read-only, or this operation is not supported on the current platform, or <i>path</i> is a directory, or the caller does not have the required permission.

The method creates the file if it doesn't exist, but it doesn't create new directories.

Therefore, the value of *path* must contain existing directories.

You can use this method to create a file that contains the following:

- The results of a [LINQ to Objects](#) query on the lines of a file, as obtained by using the `System.IO.File.ReadLines(String)` method.
- The contents of a collection that implements an `IEnumerable<T>` of strings.

5.5.2.3 AppendAllText() [1/2]

```
void ProjectGamedev.IO.File.AppendAllText (
    string path,
    string contents) [inline], [static]
```

Opens a file, appends the specified string to the file, and then closes the file.

If the file does not exist, this method creates a file, writes the specified string to the file, then closes the file.

Parameters

<i>path</i>	The file to append the specified string to.
<i>contents</i>	The string to append to the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while opening the file.
<i>UnauthorizedAccessException</i>	<i>path</i> specifies a file that is read-only, or this operation is not supported on the current platform, or <i>path</i> is a directory, or the caller does not have the required permission.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

Given a string and a file path, this method opens the specified file, appends the string to the end of the file using the specified encoding, and then closes the file.

The file handle is guaranteed to be closed by this method, even if exceptions are raised.

The method creates the file if it doesn't exist, but it doesn't create new directories.

Therefore, the value of the path parameter must contain existing directories.

5.5.2.4 AppendAllText() [2/2]

```
void ProjectGamedev.IO.File.AppendAllText (
    string path,
    string contents,
    Encoding encoding) [inline], [static]
```

Appends the specified string to the file using the specified encoding, creating the file if it does not already exist.

Parameters

<i>path</i>	The file to append the specified string to.
<i>contents</i>	The string to append to the file.
<i>encoding</i>	The character encoding to use.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	<i>path</i> is invalid (for example, the directory doesn't exist, or it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while opening the file.
<i>UnauthorizedAccessException</i>	<i>path</i> specifies a file that is read-only, or this operation is not supported on the current platform, or <i>path</i> is a directory, or the caller does not have the required permission.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

Given a string and a file path, this method opens the specified file, appends the string to the end of the file using the specified encoding, and then closes the file.

The file handle is guaranteed to be closed by this method, even if exceptions are raised.

The method creates the file if it doesn't exist, but it doesn't create new directories.

Therefore, the value of the path parameter must contain existing directories.

5.5.2.5 Copy() [1/2]

```
void ProjectGamedev.IO.File.Copy (
    string sourceFileName,
    string destFileName) [inline], [static]
```

Copies an existing file to a new file. Overwriting a file of the same name is not allowed.

Parameters

<i>sourceFileName</i>	The file to copy.
<i>destFileName</i>	The name of the destination file. This cannot be a directory.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission
<i>ArgumentException</i>	<i>sourceFileName</i> or <i>destFileName</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>sourceFileName</i> or <i>destFileName</i> specify a directory. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>sourceFileName</i> or <i>destFileName</i> is <code>null</code>
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The path specified in <i>sourceFileName</i> or <i>destFileName</i> is invalid (for example, it is on an unmapped drive).
<i>FileNotFoundException</i>	<i>sourceFileName</i> was not found.
<i>IOException</i>	<i>destFileName</i> exists or an I/O error has occurred.
<i>NotSupportedException</i>	<i>sourceFileName</i> or <i>destFileName</i> is in an invalid format.

This method is equivalent to the `Copy(String, String, Boolean)` method overload with the `overwrite` parameter set to `false`.

The *sourceFileName* and *destFileName* parameters can specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see the [Directory.GetCurrentDirectory](#) method

This method does not support wildcard characters in the parameters.

The attributes of the original file are retained in the copied file.

See also

`Directory.Move(String, String)`, `System.IO.File.Move(String, String)`

5.5.2.6 Copy() [2/2]

```
void ProjectGamedev.IO.File.Copy (
    string sourceFileName,
    string destFileName,
    bool overwrite) [inline], [static]
```

Copies an existing file to a new file. Overwriting a file of the same name is allowed.

Parameters

<i>sourceFileName</i>	The file to copy.
<i>destFileName</i>	The name of the destination file. This cannot be a directory.
<i>overwrite</i>	<code>true</code> if the destination file should be replaced if it already exists; otherwise, <code>false</code> .

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or <i>destFileName</i> is read-only, or <i>overwrite</i> is <code>true</code> , <i>destFileName</i> exists and is hidden, but <i>sourceFileName</i> is not hidden.
<i>ArgumentException</i>	<i>sourceFileName</i> or <i>destFileName</i> is a zero-length string, contains only white space, contains one or more invalid characters, or <i>sourceFileName</i> or <i>destFileName</i> specify a directory. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>sourceFileName</i> or <i>destFileName</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The path specified in <i>sourceFileName</i> or <i>destFileName</i> is invalid (for example, it is on an unmapped drive).
<i>FileNotFoundException</i>	<i>sourceFileName</i> was not found.
<i>IOException</i>	<i>destFileName</i> exists and <i>overwrite</i> is <code>false</code> , or an I/O error has occurred.
<i>NotSupportedException</i>	<i>sourceFileName</i> or <i>destFileName</i> is in an invalid format.

The *sourceFileName* and *destFileName* parameters can specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. This method does not support wildcard characters in the parameters.

The attributes of the original file are retained in the copied file.

For a list of common I/O tasks, see [Common I/O Tasks](#).

See also

`Directory.Move(String, String)`, `System.IO.File.Move(String, String)`

5.5.2.7 Create() [1/3]

```
void ProjectGamedev.IO.File.Create (
    string path) [inline], [static]
```

Creates, or truncates and overwrites, a file in the specified path.

Parameters

<i>path</i>	The path and name of the file to create.
-------------	--

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .

<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while creating the file.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

The `FileStream` object created by this method has a default `FileShare` value of `FileShare.None`; no other process or code can access the created file until the original file handle is closed.

This method is equivalent to the `Create(String, Int32)` method overload using the default buffer size of 4,096 bytes.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

If the specified file does not exist, it is created; if it does exist, and it is not read-only, the contents are deleted and overwritten.

By default, full read/write access to new files is granted to all users. The file is opened with read/write access and is closed before this method returns.

For a list of common I/O tasks, see [Common I/O tasks](#).

5.5.2.8 Create() [2/3]

```
void ProjectGamedev.IO.File.Create (
    string path,
    int bufferSize) [inline], [static]
```

Creates, or truncates and overwrites, a file in the specified path, specifying a buffer size.

Parameters

<i>path</i>	The path and name of the file to create.
<i>bufferSize</i>	The number of bytes buffered for reads and writes to the file.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while creating the file.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

The `FileStream` object created by this method has a default `FileShare` value of `FileShare.None`; no other process or code can access the created file until the original file handle is closed.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

This method is equivalent to the `FileStream(String, FileMode, FileAccess, FileShare, Int32)` constructor overload. If the specified file does not exist, it is created; if it does exist, and it is not read-only, the contents are replaced.

By default, full read/write access to new files is granted to all users. The file is opened with read/write access and is closed before this method returns.

For a list of common I/O tasks, see [Common I/O tasks](#).

5.5.2.9 Create() [3/3]

```
void ProjectGamedev.IO.File.Create (
    string path,
    int bufferSize,
    FileOptions options) [inline], [static]
```

Creates or overwrites a file in the specified path, specifying a buffer size and options that describe how to create or overwrite the file.

Parameters

<i>path</i>	The path and name of the file to create.
<i>bufferSize</i>	The number of bytes buffered for reads and writes to the file.
<i>options</i>	One of the <code>FileOptions</code> values that describes how to create or overwrite the file.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while creating the file.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

This method is equivalent to the `FileStream(String, FileMode, FileAccess, FileShare, Int32)` constructor overload. If the specified file does not exist, it is created; if it does exist, and it is not read-only, the contents are replaced.

By default, full read/write access to new files is granted to all users. The file is opened with read/write access and is closed before this method returns.

For a list of common I/O tasks, see [Common I/O tasks](#).

5.5.2.10 Delete()

```
void ProjectGamedev.IO.File.Delete (
    string path) [inline], [static]
```

Deletes the specified file.

Parameters

<i>path</i>	The name of the file to be deleted. Wildcard characters are not supported.
-------------	--

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	The specified file is in use, or there is an open handle on the file, and the operating system is Windows XP or earlier. This open handle can result from enumerating directories and files. For more information, see How to: Enumerate Directories and Files .
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or the file is an executable file that is in use, or <i>path</i> is a directory, or <i>path</i> specified a read-only file.

Specify a file name with any relative or absolute path information for the *path* parameter.

Wildcard characters cannot be included.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

If the file to be deleted does not exist, no exception is thrown.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.11 Exists()

```
bool ProjectGamedev.IO.File.Exists (
    string path) [inline], [static]
```

Determines whether the specified file exists.

Parameters

<i>path</i>	The file to check.
-------------	--------------------

Returns

`true` if the caller has the required permissions and *path* contains the name of an existing file; otherwise, `false`.

This method also returns `false` if *path* is `null`, an invalid path, or a zero-length string.

If the caller does not have sufficient permissions to read the specified file, no exception is thrown and the method returns `false` regardless of the existence of path.

Don't use the [Exists](#) method for path validation; this method merely checks if the file specified in path exists.

Passing an invalid path to [Exists](#) returns `false`.

To check whether the path contains any invalid characters, you can call the `Path.GetInvalidPathChars` method to retrieve the characters that are invalid for the file system.

You can also create a regular expression to test whether the path is valid for your environment.

For examples of acceptable paths, see `System.IO.File`.

To check if a directory exists, see [Directory.Exists](#).

Be aware that another process can potentially do something with the file in between the time you call the [Exists](#) method and perform another operation on the file, such as [Delete](#).

The *path* parameter is permitted to specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

If *path* describes a directory, this method returns `false`.

Trailing spaces are removed from the *path* parameter before determining if the file exists.

The [Exists](#) method returns `false` if any error occurs while trying to determine if the specified file exists.

This can occur in situations that raise exceptions such as passing a file name with invalid characters or too many characters, a failing or missing disk, or if the caller does not have permission to read the file.

5.5.2.12 GetAttributes()

```
FileAttributes ProjectGamedev.IO.File.GetAttributes (
    string path) [inline], [static]
```

Gets the FileAttributes of the file on the path.

Parameters

<i>path</i>	The path to the file.
-------------	-----------------------

Returns

The FileAttributes of the file on the path.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is empty, contains only white spaces, or contains invalid characters.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>FileNotFoundException</i>	<i>path</i> represents a file and is invalid, such as being on an unmapped drive, or the file cannot be found.

<i>DirectoryNotFoundException</i>	<i>path</i> represents a directory and is invalid, such as being on an unmapped drive, or the directory cannot be found.
<i>IOException</i>	This file is being used by another process.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.13 GetCreationTime()

```
DateTime ProjectGameDev.IO.File.GetCreationTime (
    string path) [inline], [static]
```

Returns the creation date and time of the specified file or directory.

Parameters

<i>path</i>	The file or directory for which to obtain creation date and time information.
-------------	---

Returns

A `DateTime` structure set to the creation date and time for the specified file or directory. This value is expressed in local time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

If the file described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC), adjusted to local time.

NTFS-formatted drives may cache information about a file, such as file creation time, for a short period of time. As a result, it may be necessary to explicitly set the creation time of a file if you are overwriting or replacing an existing file.

For a list of common I/O tasks, see [Common I/O Tasks](#).

See also

FileSystemWatcher

5.5.2.14 GetCreationTimeUtc()

```
DateTime ProjectGamedev.IO.File.GetCreationTimeUtc (
    string path) [inline], [static]
```

Returns the creation date and time, in Coordinated Universal Time (UTC), of the specified file or directory.

Parameters

<i>path</i>	The file or directory for which to obtain creation date and time information.
-------------	---

Returns

A `DateTime` structure set to the creation date and time for the specified file or directory. This value is expressed in UTC time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

If the file described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

NTFS-formatted drives may cache information about a file, such as file creation time, for a short period of time. As a result, it may be necessary to explicitly set the creation time of a file if you are overwriting or replacing an existing file.

For a list of common I/O tasks, see [Common I/O Tasks](#).

See also

`FileSystemWatcher`

5.5.2.15 GetLastAccessTime()

```
DateTime ProjectGamedev.IO.File.GetLastAccessTime (
    string path) [inline], [static]
```

Returns the date and time the specified file or directory was last accessed.

Parameters

<i>path</i>	The file or directory for which to obtain access date and time information.
-------------	---

Returns

A `DateTime` structure set to the date and time that the specified file or directory was last accessed. This value is expressed in local time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

If the file described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC), adjusted to local time.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

See also

`FileSystemWatcher`

5.5.2.16 GetLastAccessTimeUtc()

```
DateTime ProjectGamedev.IO.File.GetLastAccessTimeUtc (
    string path) [inline], [static]
```

Returns the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last accessed.

Parameters

<i>path</i>	The file or directory for which to obtain access date and time information.
-------------	---

Returns

A `DateTime` structure set to the date and time that the specified file or directory was last accessed. This value is expressed in UTC time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

If the file described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

See also

`FileSystemWatcher`

5.5.2.17 `GetLastWriteTime()`

```
DateTime ProjectGamedev.IO.File.GetLastWriteTime (
    string path) [inline], [static]
```

Returns the date and time the specified file or directory was last written to.

Parameters

<i>path</i>	The file or directory for which to obtain write date and time information.
-------------	--

Returns

A `DateTime` structure set to the date and time that the specified file or directory was last written to. This value is expressed in local time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .

<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

Each operating system manages the last write time according to its own rules.

To improve performance, an operating system might not set the last write time value to the exact time of the last write operation, but might set it to a close approximation instead.

If the file described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC), adjusted to local time.

The *path* parameter is permitted to specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

See also

FileSystemWatcher

5.5.2.18 GetLastWriteTimeUtc()

```
DateTime ProjectGamedev.IO.File.GetLastWriteTimeUtc (
    string path) [inline], [static]
```

Returns the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last written to.

Parameters

<i>path</i>	The file or directory for which to obtain write date and time information.
-------------	--

Returns

A `DateTime` structure set to the date and time that the specified file or directory was last written to. This value is expressed in UTC time.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

This method may return an inaccurate value, because it uses native functions whose values may not be continuously updated by the operating system.

Each operating system manages the last write time according to its own rules.

To improve performance, an operating system might not set the last write time value to the exact time of the last write operation, but might set it to a close approximation instead.

If the file described in the *path* parameter does not exist, this method returns 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC), adjusted to local time.

The *path* parameter is permitted to specify relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

See also

FileSystemWatcher

5.5.2.19 Move()

```
void ProjectGamedev.IO.File.Move (
    string sourceFileName,
    string destFileName) [inline], [static]
```

Moves a specified file to a new location, providing the option to specify a new file name.

Parameters

<i>sourceFileName</i>	The name of the file to move. Can include a relative or absolute path.
<i>destFileName</i>	The new path and name for the file.

Exceptions

<i>IOException</i>	<i>destFileName</i> already exists, or an I/O error has occurred, e.g. while copying the file across disk volumes.
<i>FileNotFoundException</i>	<i>sourceFileName</i> was not found.
<i>ArgumentNullException</i>	<i>sourceFileName</i> or <i>destFileName</i> is <code>null</code> .
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>sourceFileName</i> or <i>destFileName</i> is a zero-length string, contains only white space, or contains invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The path specified in <i>sourceFileName</i> or <i>destFileName</i> is invalid, (for example, it is on an unmapped drive).
<i>NotSupportedException</i>	<i>sourceFileName</i> or <i>destFileName</i> is in an invalid format.

This method works across disk volumes, and it does not throw an exception if the source and destination are the same.

Note that if you attempt to replace a file by moving a file of the same name into that directory, an `IOException` is thrown. To avoid this problem:

- In all .NET versions, you can call `Copy(String, String, Boolean)` to copy with overwrite, then call [Delete](#) to remove the excess source file.
This strategy is advisable if the file being copied is small, and you are looking for an "atomic" file operation. If you [Delete](#) the file first, and the system or program crashes, the destination file will no longer exist.
- In all .NET versions, you can call [Delete](#) before calling `Move(String, String)`, which will only delete the file if it exists.

The *sourceFileName* and *destFileName* arguments can include relative or absolute path information.

Relative path information is interpreted as relative to the current working directory.

To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

Moving the file across disk volumes is equivalent to copying the file and deleting it from the source if the copying was successful.

If you try to move a file across disk volumes and that file is in use, the file is copied to the destination, but it is not deleted from the source.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.20 ReadAllBytes()

```
byte[] ProjectGamedev.IO.File.ReadAllBytes (
    string path) [inline], [static]
```

Opens a binary file, reads the contents of the file into a byte array, and then closes the file.

Parameters

<i>path</i>	The file to open for reading.
-------------	-------------------------------

Returns

A byte array containing the contents of the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.

<i>DirectoryNotFoundException</i>	The specified path is invalid, (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while opening the file.
<i>UnauthorizedAccessException</i>	This operation is not supported on the current platform, or <i>path</i> specified a directory, or the caller does not have the required permission.
<i>FileNotFoundException</i>	The file specified in <i>path</i> was not found.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

Given a file path, this method opens the file, reads the contents of the file into a byte array, and then closes the file.

5.5.2.21 ReadAllLines() [1/2]

```
string[] ProjectGamedev.IO.File.ReadAllLines (
    string path) [inline], [static]
```

Opens a text file, reads all lines of the file, and then closes the file.

Parameters

<i>path</i>	The file to open for reading.
-------------	-------------------------------

Returns

A string array containing all lines of the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid, (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while opening the file.
<i>UnauthorizedAccessException</i>	This operation is not supported on the current platform, or <i>path</i> specified a directory, or <i>path</i> specified a file that is read-only, or the caller does not have the required permission.
<i>FileNotFoundException</i>	The file specified in <i>path</i> was not found.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

This method opens a file, reads each line of the file, then adds each line as an element of a string array. It then closes the file.

A line is defined as a sequence of characters followed by a carriage return ('\r'), a line feed ('\n'), or a carriage return immediately followed by a line feed.

The resulting string does not contain the terminating carriage return and/or line feed.

This method attempts to automatically detect the encoding of a file based on the presence of byte order marks. Encoding formats UTF-8 and UTF-32 (both big-endian and little-endian) can be detected.

5.5.2.22 ReadAllLines() [2/2]

```
string[] ProjectGamedev.IO.File.ReadAllLines (
    string path,
    Encoding encoding) [inline], [static]
```

Opens a file, reads all lines of the file with the specified encoding, and then closes the file.

Parameters

<i>path</i>	The file to open for reading.
<i>encoding</i>	The encoding applied to the contents of the file.

Returns

A string array containing all lines of the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid, (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while opening the file.
<i>UnauthorizedAccessException</i>	This operation is not supported on the current platform, or <i>path</i> specified a directory, or <i>path</i> specified a file that is read-only, or the caller does not have the required permission.
<i>FileNotFoundException</i>	The file specified in <i>path</i> was not found.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

This method opens a file, reads each line of the file, then adds each line as an element of a string array. It then closes the file.

A line is defined as a sequence of characters followed by a carriage return ('\r'), a line feed ('\n'), or a carriage return immediately followed by a line feed.

The resulting string does not contain the terminating carriage return and/or line feed.

This method attempts to automatically detect the encoding of a file based on the presence of byte order marks. Encoding formats UTF-8 and UTF-32 (both big-endian and little-endian) can be detected.

5.5.2.23 ReadAllText() [1/2]

```
string ProjectGamedev.IO.File.ReadAllText (
    string path) [inline], [static]
```

Opens a text file, reads all the text in the file, and then closes the file.

Parameters

<i>path</i>	The file to open for reading.
-------------	-------------------------------

Returns

A string containing all the text in the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid, (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while opening the file.
<i>UnauthorizedAccessException</i>	This operation is not supported on the current platform, or <i>path</i> specified a directory, or <i>path</i> specified a file that is read-only, or the caller does not have the required permission.
<i>FileNotFoundException</i>	The file specified in <i>path</i> was not found.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

This method opens a file, reads all the text in the file, and returns it as a string. It then closes the file.

This method attempts to automatically detect the encoding of a file based on the presence of byte order marks. It automatically recognizes UTF-8, little-endian UTF-16, big-endian UTF-16, little-endian UTF-32, and big-endian UTF-32 text if the file starts with the appropriate byte order marks.

Use the `ReadAllText(String, Encoding)` method overload when reading files that might contain imported text, because unrecognized characters may not be read correctly.

The file handle is guaranteed to be closed by this method, even if exceptions are raised.

5.5.2.24 ReadAllText() [2/2]

```
string ProjectGamedev.IO.File.ReadAllText (
    string path,
    Encoding encoding) [inline], [static]
```

Opens a file, reads all text in the file with the specified encoding, and then closes the file.

Parameters

<i>path</i>	The file to open for reading
<i>encoding</i>	The encoding applied to the contents of the file.

Returns

A string containing all text in the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid, (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while opening the file.
<i>UnauthorizedAccessException</i>	This operation is not supported on the current platform, or <i>path</i> specified a directory, or <i>path</i> specified a file that is read-only, or the caller does not have the required permission.
<i>FileNotFoundException</i>	The file specified in <i>path</i> was not found.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

This method opens a file, reads all the text in the file, and returns it as a string. It then closes the file.

This method attempts to automatically detect the encoding of a file based on the presence of byte order marks. It automatically recognizes UTF-8, little-endian UTF-16, big-endian UTF-16, little-endian UTF-32, and big-endian UTF-32 text if the file starts with the appropriate byte order marks.

The file handle is guaranteed to be closed by this method, even if exceptions are raised.

To use the encoding settings as configured for your operating system, specify the `Encoding.Default` property for the encoding parameter.

5.5.2.25 Replace() [1/2]

```
void ProjectGamedev.IO.File.Replace (
    string sourceFileName,
    string destinationFileName,
    string? destinationBackupFileName) [inline], [static]
```

Replaces the contents of a specified file with the contents of another file, deleting the original file and creating a backup of the replaced file.

Parameters

<i>sourceFileName</i>	The name of a file that replaces the file specified by <i>destinationFileName</i> .
<i>destinationFileName</i>	The name of the file being replaced.
<i>destinationBackupFileName</i>	The name of the backup file.

Exceptions

<i>ArgumentException</i>	The path described by the <i>destinationFileName</i> or <i>destinationBackupFileName</i> parameter is not of a legal form.
<i>ArgumentNullException</i>	The <i>destinationBackupFileName</i> is <code>null</code> .
<i>DriveNotFoundException</i>	An invalid drive was specified.

Exceptions

<i>FileNotFoundException</i>	The file described by <i>sourceFileName</i> or <i>destinationFileName</i> parameter could not be found.
<i>IOException</i>	An I/O error occurred while opening the file, or the <i>sourceFileName</i> and <i>destinationFileName</i> parameters specify the same file.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The <i>sourceFileName</i> or <i>destinationFileName</i> parameter specifies a file that is read-only, or this operation is not supported on the current platform, or source or destination parameters specify a directory instead of a file, or the caller does not have the required permission, or <i>sourceFileName</i> and <i>destinationFileName</i> specify the same existing directory.

The `Replace(String, String, String?)` method replaces the contents of a specified file with the contents of another file.

It also creates a backup of the file that was replaced.

If the *sourceFileName* and *destinationFileName* are on different volumes, this method raises an exception.

If the *destinationBackupFileName* is on a different volume from the source file, the backup file will be deleted.

Pass `null` to the *destinationBackupFileName* parameter if you don't want to create a backup of the file being replaced.

If the *destinationBackupFileName* already exists, it's replaced with the contents of the *destinationFileName* file.

5.5.2.26 Replace() [2/2]

```
void ProjectGamedev.IO.File.Replace (
    string sourceFileName,
    string destinationFileName,
    string? destinationBackupFileName,
    bool ignoreMetadataErrors) [inline], [static]
```

Replaces the contents of a specified file with the contents of another file, deleting the original file and creating a backup of the replaced file, and optionally ignores merge errors.

Parameters

<i>sourceFileName</i>	The name of a file that replaces the file specified by <i>destinationFileName</i> .
<i>destinationFileName</i>	The name of the file being replaced.
<i>destinationBackupFileName</i>	The name of the backup file.
<i>ignoreMetadataErrors</i>	<code>true</code> to ignore merge errors (such as attributes and access control lists (ACLs)) from the replaced file to the replacement file; otherwise, <code>false</code> .

Exceptions

<i>ArgumentException</i>	The path described by the <i>destinationFileName</i> or <i>destinationBackupFileName</i> parameter is not of a legal form.
<i>ArgumentNullException</i>	The <i>destinationBackupFileName</i> is <code>null</code> .

Exceptions

<i>DriveNotFoundException</i>	An invalid drive was specified.
<i>FileNotFoundException</i>	The file described by <i>sourceFileName</i> or <i>destinationFileName</i> parameter could not be found.
<i>IOException</i>	An I/O error occurred while opening the file, or the <i>sourceFileName</i> and <i>destinationFileName</i> parameters specify the same file.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>UnauthorizedAccessException</i>	The <i>sourceFileName</i> or <i>destinationFileName</i> parameter specifies a file that is read-only, or this operation is not supported on the current platform, or source or destination parameters specify a directory instead of a file, or the caller does not have the required permission, or <i>sourceFileName</i> and <i>destinationFileName</i> specify the same existing directory.

The `Replace(String, String, String?)` method replaces the contents of a specified file with the contents of another file.

It also creates a backup of the file that was replaced.

If the *sourceFileName* and *destinationFileName* are on different volumes, this method raises an exception.

If the *destinationBackupFileName* is on a different volume from the source file, the backup file will be deleted.

Pass `null` to the *destinationBackupFileName* parameter if you don't want to create a backup of the file being replaced.

If the *destinationBackupFileName* already exists, it's replaced with the contents of the *destinationFileName* file.

5.5.2.27 ReturningCriticalSection< T >()

```
T ProjectGamedev.IO.File.ReturningCriticalSection< T > (
    string path,
    Func< T > action) [inline], [static], [private]
```

A wrapper function for invoking `ConcurrentIO.CriticalSection(String, Action)` on I/O functions with a return value.

Parameters

<i>path</i>	The file/directory path.
<i>action</i>	The callable action to execute in a thread-safe manner.

Template Parameters

<i>T</i>	The type of the return value of <i>action</i> .
----------	---

Returns

The return value of *action* .

5.5.2.28 SetAttributes()

```
void ProjectGamedev.IO.File.SetAttributes (
    string path,
    FileAttributes fileAttributes) [inline], [static]
```

Sets the specified `FileAttributes` of the file on the specified path.

Parameters

<i>path</i>	The path to the file.
<i>fileAttributes</i>	A bitwise combination of the enumeration values.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is empty, contains only white spaces, contains invalid characters, or the file attribute is invalid.
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>DirectoryNotFoundException</i>	The specified path is invalid, (for example, it is on an unmapped drive).
<i>FileNotFoundException</i>	The file cannot be found.
<i>UnauthorizedAccessException</i>	<i>path</i> specified a file that is read-only, or this operation is not supported on the current platform, or <i>path</i> specified a directory, or the caller does not have the required permission.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

Certain file attributes, such as `FileAttributes.Hidden` and `FileAttributes.ReadOnly`, can be combined. Other attributes, such as `FileAttributes.Normal`, must be used alone.

It is not possible to change the compression status of a `System.IO.File` object using the `SetAttributes(String, FileAttributes)` method.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.29 SetCreationTime()

```
void ProjectGamedev.IO.File.SetCreationTime (
    string path,
    DateTime creationTime) [inline], [static]
```

Sets the date and time the file was created.

Parameters

<i>path</i>	The file for which to set the creation date and time information.
<i>creationTime</i>	A <code>DateTime</code> containing the value to set for the creation date and time of <i>path</i> . This value is expressed in local time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
------------------------------	-----------------------------------

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>ArgumentOutOfRangeException</i>	<i>creationTime</i> specifies a value outside the range of dates, times, or both permitted for this operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

NTFS-formatted drives may cache file meta-info, such as file creation time, for a short period of time. As a result, it may be necessary to explicitly set the creation time of a file if you are overwriting or replacing an existing file.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.30 SetCreationTimeUtc()

```
void ProjectGamedev.IO.File.SetCreationTimeUtc (
    string path,
    DateTime creationTimeUtc) [inline], [static]
```

Sets the date and time, in Coordinated Universal Time (UTC), that the file was created.

Parameters

<i>path</i>	The file for which to set the creation date and time information.
<i>creationTimeUtc</i>	A <code>DateTime</code> containing the value to set for the creation date and time of <i>path</i> . This value is expressed in UTC time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>IOException</i>	An I/O error occurred while performing the operation.

<i>ArgumentOutOfRangeException</i>	<i>creationTimeUtc</i> specifies a value outside the range of dates, times, or both permitted for this operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

NTFS-formatted drives may cache file meta-info, such as file creation time, for a short period of time. As a result, it may be necessary to explicitly set the creation time of a file if you are overwriting or replacing an existing file.

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.31 SetLastAccessTime()

```
void ProjectGamedev.IO.File.SetLastAccessTime (
    string path,
    DateTime lastAccessTime) [inline], [static]
```

Sets the date and time the specified file was last accessed.

Parameters

<i>path</i>	The file for which to set the access date and time information.
<i>lastAccessTime</i>	A <i>DateTime</i> containing the value to set for the last access date and time of <i>path</i> . This value is expressed in local time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <i>Path.GetInvalidPathChars</i> method.
<i>ArgumentNullException</i>	<i>path</i> is <i>null</i> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>ArgumentOutOfRangeException</i>	<i>lastAccessTime</i> specifies a value outside the range of dates, times, or both permitted for this operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.32 SetLastAccessTimeUtc()

```
void ProjectGamedev.IO.File.SetLastAccessTimeUtc (
    string path,
    DateTime lastAccessTimeUtc) [inline], [static]
```

Sets the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last accessed.

Parameters

<i>path</i>	The file for which to set the access date and time information.
<i>lastAccessTimeUtc</i>	A <i>DateTime</i> containing the value to set for the last access date and time of <i>path</i> . This value is expressed in UTC time.

Exceptions

<i>FileNotFoundException</i>	The specified path was not found.
<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <i>Path.GetInvalidPathChars</i> method.
<i>ArgumentNullException</i>	<i>path</i> is <i>null</i> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>ArgumentOutOfRangeException</i>	<i>lastAccessTimeUtc</i> specifies a value outside the range of dates, times, or both permitted for this operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.33 SetLastWriteTime()

```
void ProjectGamedev.IO.File.SetLastWriteTime (
    string path,
    DateTime lastWriteTime) [inline], [static]
```

Sets the date and time that the specified file was last written to.

Parameters

<i>path</i>	The file for which to set the date and time information.
<i>lastWriteTime</i>	A <i>DateTime</i> containing the value to set for the last write date and time of <i>path</i> . This value is expressed in local time.

Exceptions

<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> ,
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>ArgumentOutOfRangeException</i>	<i>lastWriteTime</i> specifies a value outside the range of dates, times, or both permitted for this operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.34 SetLastWriteTimeUtc()

```
void ProjectGamedev.IO.File.SetLastWriteTimeUtc (
    string path,
    DateTime lastWriteTimeUtc) [inline], [static]
```

Sets the date and time, in Coordinated Universal Time (UTC), that the specified file or directory was last written to.

Parameters

<i>path</i>	The file for which to set the date and time information.
<i>lastWriteTimeUtc</i>	A <code>DateTime</code> containing the value to set for the last write date and time of <i>path</i> . This value is expressed in UTC time.

Exceptions

<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> ,
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>ArgumentOutOfRangeException</i>	<i>lastWriteTimeUtc</i> specifies a value outside the range of dates, times, or both permitted for this operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission.

The *path* parameter is permitted to specify relative or absolute path information. Relative path information is interpreted as relative to the current working directory. To obtain the current working directory, see [Directory.GetCurrentDirectory](#).

For a list of common I/O tasks, see [Common I/O Tasks](#).

5.5.2.35 WriteAllBytes()

```
void ProjectGamedev.IO.File.WriteAllBytes (
    string path,
    byte[] bytes) [inline], [static]
```

Creates a new file, writes the specified byte array to the file, and then closes the file. If the target file already exists, it is truncated and overwritten.

Parameters

<i>path</i>	The file to write to.
<i>bytes</i>	The bytes to write to the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or this operation is not supported on the current platform, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden, or <i>path</i> specified a directory.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

Given a byte array and a file path, this method opens the specified file, writes the contents of the byte array to the file, and then closes the file.

5.5.2.36 WriteAllLines() [1/4]

```
void ProjectGamedev.IO.File.WriteAllLines (
    string path,
    IEnumerable< string > contents) [inline], [static]
```

Creates a new file, writes a collection of strings to the file, and then closes the file.

Parameters

<i>path</i>	The file to write to.
<i>contents</i>	The lines to write to the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> or <i>contents</i> is <code>null</code> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while performing the operation.

Exceptions

<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or this operation is not supported on the current platform, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden, or <i>path</i> specified a directory.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

The default behavior of the `WriteAllLines(String, IEnumerable<String>)` method is to write out data by using UTF-8 encoding without a byte order mark (BOM).

If it is necessary to include a UTF-8 identifier, such as a byte order mark, at the beginning of a file, use the `WriteAllLines(String, IEnumerable<String>, Encoding)` method overload with `Encoding.UTF8` encoding.

If the target file already exists, it is truncated and overwritten.

You can use this method to create the contents for a collection class that takes an `IEnumerable<T>` in its constructor, such as a `List<T>`, `HashSet<T>`, or a `SortedSet<T>` class.

5.5.2.37 WriteAllLines() [2/4]

```
void ProjectGamedev.IO.File.WriteAllLines (
    string path,
    IEnumerable< string > contents,
    Encoding encoding) [inline], [static]
```

Creates a new file by using the specified encoding, writes a collection of strings to the file, and then closes the file.

Parameters

<i>path</i>	The file to write to.
<i>contents</i>	The lines to write to the file.
<i>encoding</i>	The character encoding to use.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> , <i>contents</i> , or <i>encoding</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or this operation is not supported on the current platform, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden, or <i>path</i> specified a directory.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.

<i>SecurityException</i>	The caller does not have the required permission.
--------------------------	---

If the target file already exists, it is truncated and overwritten.

You can use this method to create a file that contains the following:

- The results of a [LINQ to Objects](#) query on the lines of a file, as obtained by using the `System.IO.File.ReadLines(String, Encoding)` method.
- The contents of a collection that implements an `IEnumerable<T>` of strings.

5.5.2.38 WriteAllLines() [3/4]

```
void ProjectGamedev.IO.File.WriteAllLines (
    string path,
    string[] contents) [inline], [static]
```

Creates a new file, write the specified string array to the file, and then closes the file.

Parameters

<i>path</i>	The file to write to.
<i>contents</i>	The string array to write to the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the <code>Path.GetInvalidPathChars</code> method.
<i>ArgumentNullException</i>	<i>path</i> or <i>contents</i> is <code>null</code> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or this operation is not supported on the current platform, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden, or <i>path</i> specified a directory.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

If the target file already exists, it is truncated and overwritten.

The default behavior of the `WriteAllLines(String, IEnumerable<String>)` method is to write out data by using UTF-8 encoding without a byte order mark (BOM).

If it is necessary to include a UTF-8 identifier, such as a byte order mark, at the beginning of a file, use the `WriteAllLines(String, String[], Encoding)` method overload with `Encoding.UTF8` encoding.

Given a string array and a file path, this method opens the specified file, writes the string array to the file, and then closes the file.

5.5.2.39 WriteAllLines() [4/4]

```
void ProjectGamedev.IO.File.WriteAllLines (
    string path,
    string[] contents,
    Encoding encoding) [inline], [static]
```

Creates a new file, writes the specified string array to the file by using the specified encoding, and then closes the file.

Parameters

<i>path</i>	The file to write to.
<i>contents</i>	The string array to write to the file.
<i>encoding</i>	An Encoding object that represents the character encoding applied to the string array.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> or <i>contents</i> is <code>null</code> ,
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or this operation is not supported on the current platform, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden, or <i>path</i> specified a directory.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

If the target file already exists, it is truncated and overwritten.

Given a string array and a file path, this method opens the specified file, writes the string array to the file using the specified encoding, and then closes the file.

5.5.2.40 WriteAllText() [1/2]

```
void ProjectGamedev.IO.File.WriteAllText (
    string path,
    string contents) [inline], [static]
```

Creates a new file, writes the specified string to the file, and then closes the file.

If the target file already exists, it is truncated and overwritten.

Parameters

<i>path</i>	The file to write to.
<i>contents</i>	The string to write to the file.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.
<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or this operation is not supported on the current platform, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden, or <i>path</i> specified a directory.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

This method uses UTF-8 encoding without a Byte-Order Mark (BOM), so using the Encoding.GetPreamble method will return an empty byte array.

If it is necessary to include a UTF-8 identifier, such as a byte order mark, at the beginning of a file, use the WriteAllText(String, String, Encoding) method overload with UTF8 encoding.

Given a string and a file path, this method opens the specified file, writes the string to the file, and then closes the file.

5.5.2.41 WriteAllText() [2/2]

```
void ProjectGamedev.IO.File.WriteAllText (
    string path,
    string contents,
    Encoding encoding) [inline], [static]
```

Creates a new file, writes the specified string to the file using the specified encoding, and then closes the file. If the target file already exists, it is truncated and overwritten.

Parameters

<i>path</i>	The file to write to.
<i>contents</i>	The string to write to the file.
<i>encoding</i>	The encoding to apply to the string.

Exceptions

<i>ArgumentException</i>	.NET Framework and .NET Core versions older than 2.1: <i>path</i> is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the Path.GetInvalidPathChars method.
<i>ArgumentNullException</i>	<i>path</i> is <code>null</code> .
<i>PathTooLongException</i>	The specified path, file name, or both exceed the system-defined maximum length.

Exceptions

<i>DirectoryNotFoundException</i>	The specified path is invalid (for example, it is on an unmapped drive).
<i>IOException</i>	An I/O error occurred while performing the operation.
<i>UnauthorizedAccessException</i>	The caller does not have the required permission, or this operation is not supported on the current platform, or <i>path</i> specified a file that is read-only, or <i>path</i> specified a file that is hidden, or <i>path</i> specified a directory.
<i>NotSupportedException</i>	<i>path</i> is in an invalid format.
<i>SecurityException</i>	The caller does not have the required permission.

Given a string and a file path, this method opens the specified file, writes the string to the file using the specified encoding, and then closes the file.

The file handle is guaranteed to be closed by this method, even if exceptions are raised.

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/[File.cs](#)

5.6 ProjectGamedev.IO.Tests.PlayModeTests.FileTest Class Reference

Test class containing test cases for the [File](#) class.

Public Member Functions

- void [BeforeAll](#) ()
- void [TestFileMethodsNoTimeoutsNoExceptions](#) ()

Test ID: IO-6.

Private Member Functions

- void [CallAllFileMethods](#) ()

Private Attributes

- string [testfilepath1](#)
- string [testfilepath2](#)
- string [testfilepath3](#)
- string [testfilepath4](#)
- string [testdirectorypath1](#)
- string [testdirectorypath2](#)

5.6.1 Detailed Description

Test class containing test cases for the [File](#) class.

5.6.2 Member Function Documentation

5.6.2.1 BeforeAll()

```
void ProjectGamedev.IO.Tests.PlayModeTests.FileTest.BeforeAll () [inline]
```

5.6.2.2 CallAllFileMethods()

```
void ProjectGamedev.IO.Tests.PlayModeTests.FileTest.CallAllFileMethods () [inline], [private]
```

5.6.2.3 TestFileMethodsNoTimeoutsNoExceptions()

```
void ProjectGamedev.IO.Tests.PlayModeTests.FileTest.TestFileMethodsNoTimeoutsNoExceptions ()  
[inline]
```

Test ID: IO-6.

All supported methods of the System.IO.File class are called together to assert no deadlocks, timeouts, or other [IO](#) exceptions occur.

5.6.3 Member Data Documentation

5.6.3.1 testdirectorypath1

```
string ProjectGamedev.IO.Tests.PlayModeTests.FileTest.testdirectorypath1 [private]
```

5.6.3.2 testdirectorypath2

```
string ProjectGamedev.IO.Tests.PlayModeTests.FileTest.testdirectorypath2 [private]
```

5.6.3.3 testfilepath1

```
string ProjectGamedev.IO.Tests.PlayModeTests.FileTest.testfilepath1 [private]
```

5.6.3.4 testfilepath2

```
string ProjectGamedev.IO.Tests.PlayModeTests.FileTest.testfilepath2 [private]
```

5.6.3.5 testfilepath3

```
string ProjectGamedev.IO.Tests.PlayModeTests.FileTest.testfilepath3 [private]
```

5.6.3.6 testfilepath4

```
string ProjectGamedev.IO.Tests.PlayModeTests.FileTest.testfilepath4 [private]
```

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/[FileTest.cs](#)

5.7 ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings Class Reference

SetUpFixture class for the tests. Defines global pre- and post-test setup and teardown.

Public Member Functions

- void [GlobalSetUp](#) ()
- void [GlobalTeardown](#) ()

5.7.1 Detailed Description

SetUpFixture class for the tests. Defines global pre- and post-test setup and teardown.

5.7.2 Member Function Documentation

5.7.2.1 GlobalSetUp()

```
void ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings.GlobalSetUp () [inline]
```

5.7.2.2 GlobalTeardown()

```
void ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings.GlobalTeardown () [inline]
```

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/[GlobalTestSettings.cs](#)

5.8 ProjectGamedev.IO.ConcurrentIO.LockObject Class Reference

An empty class. Lock objects created from this class can be used with C#'s `lock` keyword for locking.

5.8.1 Detailed Description

An empty class. Lock objects created from this class can be used with C#'s `lock` keyword for locking.

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/[ConcurrentIO.cs](#)

5.9 ProjectGamedev.IO.Tests.TestConstants Class Reference

Container for constants used throughout the tests.

Static Public Attributes

- const int [TEST_THREAD_COUNT](#) = 1024
- const int [TIMEOUT_MILLISECONDS](#) = 30000
- const int [TIMEOUT_LARGE_MILLISECONDS](#) = 120000
- const string [TEST_ROOT_FOLDER](#) = "IO"
- const string [TEST_FILE_NAME_1](#) = "CONCURRENT_IO_TEST_FILE_1.tmp"
- const string [TEST_FILE_NAME_2](#) = "CONCURRENT_IO_TEST_FILE_2.tmp"
- const string [TEST_FILE_NAME_7](#) = "CONCURRENT_IO_TEST_FILE_3.tmp"
- const string [TEST_FILE_NAME_3](#) = "FILE_TEST_1.tmp"
- const string [TEST_FILE_NAME_4](#) = "FILE_TEST_2.tmp"
- const string [TEST_FILE_NAME_8](#) = "FILE_TEST_3.tmp"
- const string [TEST_FILE_NAME_9](#) = "FILE_TEST_4.tmp"
- const string [TEST_FILE_NAME_5](#) = "DIRECTORY_TEST_1.tmp"
- const string [TEST_FILE_NAME_6](#) = "DIRECTORY_TEST_2.tmp"
- const string [TEST_DIRECTORY_NAME_1](#) = "CONCURRENT_IO_TEST_TEMP_DIR"
- const string [TEST_DIRECTORY_NAME_2](#) = "CONCURRENT_IO_TEST_TEMP_DIR_COPY"
- const string [TEST_DIRECTORY_NAME_3](#) = "FILE_TEST_DIR_1"
- const string [TEST_DIRECTORY_NAME_4](#) = "FILE_TEST_DIR_2"
- const string [TEST_DIRECTORY_NAME_5](#) = "DIRECTORY_TEST_DIR_1"
- const string [TEST_DIRECTORY_NAME_6](#) = "DIRECTORY_TEST_DIR_2"

5.9.1 Detailed Description

Container for constants used throughout the tests.

5.9.2 Member Data Documentation

5.9.2.1 TEST_DIRECTORY_NAME_1

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_DIRECTORY_NAME_1 = "CONCURRENT_IO_↵
TEST_TEMP_DIR" [static]
```

5.9.2.2 TEST_DIRECTORY_NAME_2

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_DIRECTORY_NAME_2 = "CONCURRENT_IO_↵  
TEST_TEMP_DIR_COPY" [static]
```

5.9.2.3 TEST_DIRECTORY_NAME_3

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_DIRECTORY_NAME_3 = "FILE_TEST_DIR_1"  
[static]
```

5.9.2.4 TEST_DIRECTORY_NAME_4

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_DIRECTORY_NAME_4 = "FILE_TEST_DIR_2"  
[static]
```

5.9.2.5 TEST_DIRECTORY_NAME_5

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_DIRECTORY_NAME_5 = "DIRECTORY_TEST_↵  
DIR_1" [static]
```

5.9.2.6 TEST_DIRECTORY_NAME_6

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_DIRECTORY_NAME_6 = "DIRECTORY_TEST_↵  
DIR_2" [static]
```

5.9.2.7 TEST_FILE_NAME_1

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_1 = "CONCURRENT_IO_TEST_↵  
FILE_1.tmp" [static]
```

5.9.2.8 TEST_FILE_NAME_2

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_2 = "CONCURRENT_IO_TEST_↵  
FILE_2.tmp" [static]
```

5.9.2.9 TEST_FILE_NAME_3

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_3 = "FILE_TEST_1.tmp" [static]
```

5.9.2.10 TEST_FILE_NAME_4

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_4 = "FILE_TEST_2.tmp" [static]
```

5.9.2.11 TEST_FILE_NAME_5

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_5 = "DIRECTORY_TEST_1.tmp"
[static]
```

5.9.2.12 TEST_FILE_NAME_6

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_6 = "DIRECTORY_TEST_2.tmp"
[static]
```

5.9.2.13 TEST_FILE_NAME_7

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_7 = "CONCURRENT_IO_TEST_↵
FILE_3.tmp" [static]
```

5.9.2.14 TEST_FILE_NAME_8

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_8 = "FILE_TEST_3.tmp" [static]
```

5.9.2.15 TEST_FILE_NAME_9

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_FILE_NAME_9 = "FILE_TEST_4.tmp" [static]
```

5.9.2.16 TEST_ROOT_FOLDER

```
const string ProjectGamedev.IO.Tests.TestConstants.TEST_ROOT_FOLDER = "IO" [static]
```

5.9.2.17 TEST_THREAD_COUNT

```
const int ProjectGamedev.IO.Tests.TestConstants.TEST_THREAD_COUNT = 1024 [static]
```

5.9.2.18 TIMEOUT_LARGE_MILLISECONDS

```
const int ProjectGamedev.IO.Tests.TestConstants.TIMEOUT_LARGE_MILLISECONDS = 120000 [static]
```

5.9.2.19 TIMEOUT_MILLISECONDS

```
const int ProjectGamedev.IO.Tests.TestConstants.TIMEOUT_MILLISECONDS = 30000 [static]
```

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/[TestConstants.cs](#)

5.10 ProjectGamedev.IO.Tests.TestFactory Class Reference

This class is used in test setup phases (BeforeAll/AfterAll/etc) to create and delete the necessary test data (e.g. directories and files).

Static Public Member Functions

- static string [CreateTestDirectory](#) (string dirName)
Creates an empty directory named `dirName` under the test root location defined in [TestFactory](#).
- static string [CreateTestFile](#) (string dirPath, string fileName)
Creates an empty file named `fileName` inside folder `dirPath` .
- static void [DeleteTestRootDirectory](#) ()
Delete the root test directory, along with any subdirectories and files.

Static Public Attributes

- static string [Root](#)
Root test directory. Set at runtime by the `GlobalTestSettings` class.

5.10.1 Detailed Description

This class is used in test setup phases (BeforeAll/AfterAll/etc) to create and delete the necessary test data (e.g. directories and files).

5.10.2 Member Function Documentation

5.10.2.1 CreateTestDirectory()

```
string ProjectGamedev.IO.Tests.TestFactory.CreateTestDirectory (
    string dirName) [inline], [static]
```

Creates an empty directory named *dirName* under the test root location defined in [TestFactory](#).

Parameters

<i>dirName</i>	The name (not path!) of the created test directory.
----------------	---

Returns

The full path to the test directory, including its name.

5.10.2.2 CreateTestFile()

```
string ProjectGamedev.IO.Tests.TestFactory.CreateTestFile (
    string dirPath,
    string fileName) [inline], [static]
```

Creates an empty file named *fileName* inside folder *dirPath* .

Parameters

<i>dirPath</i>	The full path of the directory where the file will be located.
<i>fileName</i>	The name (not path!) of the file to be created.

Returns

The full path to the new test file, including its name.

5.10.2.3 DeleteTestRootDirectory()

```
void ProjectGamedev.IO.Tests.TestFactory.DeleteTestRootDirectory () [inline], [static]
```

Delete the root test directory, along with any subdirectories and files.

5.10.3 Member Data Documentation

5.10.3.1 Root

```
string ProjectGamedev.IO.Tests.TestFactory.Root [static]
```

Root test directory. Set at runtime by the GlobalTestSettings class.

The documentation for this class was generated from the following file:

- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/[TestFactory.cs](#)

Chapter 6

File Documentation

6.1 Assets/Project GAMEDEV/Tools/IO/Scripts/ConcurrentIO.cs File Reference

Classes

- class [ProjectGameDev.IO.ConcurrentIO](#)
*A simple concurrent wrapper for performing [IO](#) operations in parallel.
This class uses named mutex locks to ensure 2 threads do not simultaneously operate on the same file path, and to also allow several threads to simultaneously operate on different file or directory paths.*
- class [ProjectGameDev.IO.ConcurrentIO.LockObject](#)
An empty class. Lock objects created from this class can be used with C#'s `lock` keyword for locking.

Namespaces

- namespace [ProjectGameDev](#)
- namespace [ProjectGameDev.IO](#)

6.2 Assets/Project GAMEDEV/Tools/IO/Scripts/Directory.cs File Reference

Classes

- class [ProjectGameDev.IO.Directory](#)
A lightweight thread-safe directory utility serving as an abstraction over the raw `System.IO.Directory` class. Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#). Does not handle exceptions.

Namespaces

- namespace [ProjectGameDev](#)
- namespace [ProjectGameDev.IO](#)

6.3 Assets/Project GAMEDEV/Tools/IO/Scripts/File.cs File Reference

Classes

- class [ProjectGamedev.IO.File](#)

*A lightweight thread-safe file utility serving as an abstraction over the raw `System.IO.File` class.
Implements methods of the above class in a thread-safe and optimized manner using [ConcurrentIO](#).
Does not handle exceptions.*

Namespaces

- namespace [ProjectGamedev](#)
- namespace [ProjectGamedev.IO](#)

6.4 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/ConcurrentIOTest.cs File Reference

Classes

- class [ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest](#)

Test class containing test cases for the [ConcurrentIO](#) class.

Namespaces

- namespace [ProjectGamedev](#)
- namespace [ProjectGamedev.IO](#)
- namespace [ProjectGamedev.IO.Tests](#)
- namespace [ProjectGamedev.IO.Tests.PlayModeTests](#)

6.5 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/DirectoryTest.cs File Reference

Classes

- class [ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest](#)

Test class containing test cases for the [Directory](#) class.

Namespaces

- namespace [ProjectGamedev](#)
- namespace [ProjectGamedev.IO](#)
- namespace [ProjectGamedev.IO.Tests](#)
- namespace [ProjectGamedev.IO.Tests.PlayModeTests](#)

6.6 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/FileTest.cs File Reference

Classes

- class [ProjectGamedev.IO.Tests.PlayModeTests.FileTest](#)
Test class containing test cases for the [File](#) class.

Namespaces

- namespace [ProjectGamedev](#)
- namespace [ProjectGamedev.IO](#)
- namespace [ProjectGamedev.IO.Tests](#)
- namespace [ProjectGamedev.IO.Tests.PlayModeTests](#)

6.7 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/GlobalTestSettings.cs File Reference

Classes

- class [ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings](#)
SetUpFixture class for the tests. Defines global pre- and post-test setup and teardown.

Namespaces

- namespace [ProjectGamedev](#)
- namespace [ProjectGamedev.IO](#)
- namespace [ProjectGamedev.IO.Tests](#)
- namespace [ProjectGamedev.IO.Tests.PlayModeTests](#)

6.8 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/TestConstants.cs File Reference

Classes

- class [ProjectGamedev.IO.Tests.TestConstants](#)
Container for constants used throughout the tests.

Namespaces

- namespace [ProjectGamedev](#)
- namespace [ProjectGamedev.IO](#)
- namespace [ProjectGamedev.IO.Tests](#)

6.9 Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/TestFactory.cs File Reference

Classes

- class [ProjectGamedev.IO.Tests.TestFactory](#)

This class is used in test setup phases (BeforeAll/AfterAll/etc) to create and delete the necessary test data (e.g. directories and files).

Namespaces

- namespace [ProjectGamedev](#)
- namespace [ProjectGamedev.IO](#)
- namespace [ProjectGamedev.IO.Tests](#)

Index

- AppendAllLines
 - ProjectGamedev.IO.File, [69](#), [70](#)
- AppendAllText
 - ProjectGamedev.IO.File, [71](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/ConcurrentIO.cs, [111](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/Directory.cs, [111](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/File.cs, [112](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/ConcurrentIOTest.cs, [112](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/DirectoryTest.cs, [112](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/FileTest.cs, [113](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/Play Mode Tests/GlobalTestSettings.cs, [113](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/TestConstants.cs, [113](#)
- Assets/Project GAMEDEV/Tools/IO/Scripts/Tests/TestFactory.cs, [114](#)
- BeforeAll
 - ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [13](#)
 - ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, [66](#)
 - ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)
- CallAllDirectoryMethods
 - ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, [66](#)
- CallAllFileMethods
 - ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)
- Copy
 - ProjectGamedev.IO.File, [72](#), [73](#)
- CopyDirectory
 - ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [13](#)
- Create
 - ProjectGamedev.IO.File, [74–76](#)
- CreateDirectory
 - ProjectGamedev.IO.Directory, [17](#)
- CreateTestDirectory
 - ProjectGamedev.IO.Tests.TestFactory, [108](#)
- CreateTestFile
 - ProjectGamedev.IO.Tests.TestFactory, [108](#)
- CriticalSection
 - ProjectGamedev.IO.ConcurrentIO, [10](#)
- Delete
 - ProjectGamedev.IO.Directory, [18](#), [19](#)
 - ProjectGamedev.IO.File, [76](#)
- DeleteTestRootDirectory
 - ProjectGamedev.IO.Tests.TestFactory, [109](#)
- EnumerateDirectories
 - ProjectGamedev.IO.Directory, [20–22](#), [24](#)
- EnumerateFiles
 - ProjectGamedev.IO.Directory, [25–27](#), [29](#)
- EnumerateFileSystemEntries
 - ProjectGamedev.IO.Directory, [30–32](#), [34](#)
- ExclusiveSection
 - ProjectGamedev.IO.ConcurrentIO, [11](#)
- Exists
 - ProjectGamedev.IO.Directory, [35](#)
 - ProjectGamedev.IO.File, [77](#)
- GetAttributes
 - ProjectGamedev.IO.File, [78](#)
- GetCreationTime
 - ProjectGamedev.IO.Directory, [36](#)
 - ProjectGamedev.IO.File, [79](#)
- GetCreationTimeUtc
 - ProjectGamedev.IO.Directory, [37](#)
 - ProjectGamedev.IO.File, [79](#)
- GetCurrentDirectory
 - ProjectGamedev.IO.Directory, [37](#)
- GetDirectories
 - ProjectGamedev.IO.Directory, [38–40](#), [42](#)
- GetDirectoryRoot
 - ProjectGamedev.IO.Directory, [43](#)
- GetFiles
 - ProjectGamedev.IO.Directory, [44–46](#), [48](#)
- GetFileSystemEntries
 - ProjectGamedev.IO.Directory, [49–51](#), [53](#)
- GetLastAccessTime
 - ProjectGamedev.IO.Directory, [54](#)
 - ProjectGamedev.IO.File, [80](#)
- GetLastAccessTimeUtc
 - ProjectGamedev.IO.Directory, [55](#)
 - ProjectGamedev.IO.File, [81](#)
- GetLastWriteTime
 - ProjectGamedev.IO.Directory, [56](#)
 - ProjectGamedev.IO.File, [82](#)
- GetLastWriteTimeUtc
 - ProjectGamedev.IO.Directory, [57](#)
 - ProjectGamedev.IO.File, [83](#)

- GetLogicalDrives
 - ProjectGamedev.IO.Directory, 57
- GetParent
 - ProjectGamedev.IO.Directory, 58
- GlobalSetUp
 - ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings, 104
- GlobalTeardown
 - ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSettings, 104
- locks
 - ProjectGamedev.IO.ConcurrentIO, 11
- MAX_LOCK_COUNT
 - ProjectGamedev.IO.ConcurrentIO, 11
- Move
 - ProjectGamedev.IO.Directory, 59
 - ProjectGamedev.IO.File, 84
- ProjectGamedev, 7
- ProjectGamedev.IO, 7
- ProjectGamedev.IO.ConcurrentIO, 9
 - CriticalSection, 10
 - ExclusiveSection, 11
 - locks, 11
 - MAX_LOCK_COUNT, 11
 - PruneLocks, 11
 - readerWriterLock, 12
- ProjectGamedev.IO.ConcurrentIO.LockObject, 104
- ProjectGamedev.IO.Directory, 15
 - CreateDirectory, 17
 - Delete, 18, 19
 - EnumerateDirectories, 20–22, 24
 - EnumerateFiles, 25–27, 29
 - EnumerateFileSystemEntries, 30–32, 34
 - Exists, 35
 - GetCreationTime, 36
 - GetCreationTimeUtc, 37
 - GetCurrentDirectory, 37
 - GetDirectories, 38–40, 42
 - GetDirectoryRoot, 43
 - GetFiles, 44–46, 48
 - GetFileSystemEntries, 49–51, 53
 - GetLastAccessTime, 54
 - GetLastAccessTimeUtc, 55
 - GetLastWriteTime, 56
 - GetLastWriteTimeUtc, 57
 - GetLogicalDrives, 57
 - GetParent, 58
 - Move, 59
 - ReturningExclusiveSection< T >, 60
 - SetCreationTime, 60
 - SetCreationTimeUtc, 61
 - SetCurrentDirectory, 62
 - SetLastAccessTime, 62
 - SetLastAccessTimeUtc, 63
 - SetLastWriteTime, 64
 - SetLastWriteTimeUtc, 65
- ProjectGamedev.IO.File, 67
 - AppendAllLines, 69, 70
 - AppendAllText, 71
 - Copy, 72, 73
 - Create, 74–76
 - Delete, 76
 - Exists, 77
 - GetAttributes, 78
 - GetCreationTime, 79
 - GetCreationTimeUtc, 79
 - GetLastAccessTime, 80
 - GetLastAccessTimeUtc, 81
 - GetLastWriteTime, 82
 - GetLastWriteTimeUtc, 83
 - Move, 84
 - ReadAllBytes, 85
 - ReadAllLines, 86
 - ReadAllText, 87, 88
 - Replace, 89, 90
 - ReturningCriticalSection< T >, 91
 - SetAttributes, 91
 - SetCreationTime, 92
 - SetCreationTimeUtc, 93
 - SetLastAccessTime, 94
 - SetLastAccessTimeUtc, 94
 - SetLastWriteTime, 95
 - SetLastWriteTimeUtc, 96
 - WriteAllBytes, 96
 - WriteAllLines, 97–99
 - WriteAllText, 100, 101
- ProjectGamedev.IO.Tests, 7
- ProjectGamedev.IO.Tests.PlayModeTests, 8
- ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, 12
 - BeforeAll, 13
 - CopyDirectory, 13
 - TestConcurrentAccessToFile, 13
 - TestCriticalSectionParameterChecks, 13
 - testdirectorypath1, 14
 - testdirectorypath2, 14
 - TestDualKeyLocking, 13
 - TestExclusiveLocking, 14
 - TestExclusiveLockingNoTimeout, 14
 - TestExclusiveSectionParameterChecks, 14
 - testfilepath1, 14
 - testfilepath2, 14
 - testfilepath3, 15
- ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, 66
 - BeforeAll, 66
 - CallAllDirectoryMethods, 66
 - TestDirectoryMethodsNoTimeoutsNoExceptions, 66
 - testdirectorypath1, 67
 - testdirectorypath2, 67
 - testfilepath1, 67
 - testfilepath2, 67
- ProjectGamedev.IO.Tests.PlayModeTests.FileTest, 102

- BeforeAll, [103](#)
- CallAllFileMethods, [103](#)
- testdirectorypath1, [103](#)
- testdirectorypath2, [103](#)
- TestFileMethodsNoTimeoutsNoExceptions, [103](#)
- testfilepath1, [103](#)
- testfilepath2, [103](#)
- testfilepath3, [103](#)
- testfilepath4, [103](#)
- ProjectGamedev.IO.Tests.PlayModeTests.GlobalTestSetup, [104](#)
 - GlobalSetUp, [104](#)
 - GlobalTeardown, [104](#)
- ProjectGamedev.IO.Tests.TestConstants, [105](#)
 - TEST_DIRECTORY_NAME_1, [105](#)
 - TEST_DIRECTORY_NAME_2, [105](#)
 - TEST_DIRECTORY_NAME_3, [106](#)
 - TEST_DIRECTORY_NAME_4, [106](#)
 - TEST_DIRECTORY_NAME_5, [106](#)
 - TEST_DIRECTORY_NAME_6, [106](#)
 - TEST_FILE_NAME_1, [106](#)
 - TEST_FILE_NAME_2, [106](#)
 - TEST_FILE_NAME_3, [106](#)
 - TEST_FILE_NAME_4, [106](#)
 - TEST_FILE_NAME_5, [106](#)
 - TEST_FILE_NAME_6, [107](#)
 - TEST_FILE_NAME_7, [107](#)
 - TEST_FILE_NAME_8, [107](#)
 - TEST_FILE_NAME_9, [107](#)
 - TEST_ROOT_FOLDER, [107](#)
 - TEST_THREAD_COUNT, [107](#)
 - TIMEOUT_LARGE_MILLISECONDS, [107](#)
 - TIMEOUT_MILLISECONDS, [107](#)
- ProjectGamedev.IO.Tests.TestFactory, [108](#)
 - CreateTestDirectory, [108](#)
 - CreateTestFile, [108](#)
 - DeleteTestRootDirectory, [109](#)
 - Root, [109](#)
- PruneLocks
 - ProjectGamedev.IO.ConcurrentIO, [11](#)
- ReadAllBytes
 - ProjectGamedev.IO.File, [85](#)
- ReadAllLines
 - ProjectGamedev.IO.File, [86](#)
- ReadAllText
 - ProjectGamedev.IO.File, [87](#), [88](#)
- readerWriterLock
 - ProjectGamedev.IO.ConcurrentIO, [12](#)
- Replace
 - ProjectGamedev.IO.File, [89](#), [90](#)
- ReturningCriticalSection< T >
 - ProjectGamedev.IO.File, [91](#)
- ReturningExclusiveSection< T >
 - ProjectGamedev.IO.Directory, [60](#)
- Root
 - ProjectGamedev.IO.Tests.TestFactory, [109](#)
- SetAttributes
 - ProjectGamedev.IO.File, [91](#)
- SetCreationTime
 - ProjectGamedev.IO.Directory, [60](#)
 - ProjectGamedev.IO.File, [92](#)
- SetCreationTimeUtc
 - ProjectGamedev.IO.Directory, [61](#)
 - ProjectGamedev.IO.File, [93](#)
- SetCurrentDirectory
 - ProjectGamedev.IO.Directory, [62](#)
- SetLastAccessTime
 - ProjectGamedev.IO.Directory, [62](#)
 - ProjectGamedev.IO.File, [94](#)
- SetLastAccessTimeUtc
 - ProjectGamedev.IO.Directory, [63](#)
 - ProjectGamedev.IO.File, [94](#)
- SetLastWriteTime
 - ProjectGamedev.IO.Directory, [64](#)
 - ProjectGamedev.IO.File, [95](#)
- SetLastWriteTimeUtc
 - ProjectGamedev.IO.Directory, [65](#)
 - ProjectGamedev.IO.File, [96](#)
- TEST_DIRECTORY_NAME_1
 - ProjectGamedev.IO.Tests.TestConstants, [105](#)
- TEST_DIRECTORY_NAME_2
 - ProjectGamedev.IO.Tests.TestConstants, [105](#)
- TEST_DIRECTORY_NAME_3
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_DIRECTORY_NAME_4
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_DIRECTORY_NAME_5
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_DIRECTORY_NAME_6
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_FILE_NAME_1
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_FILE_NAME_2
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_FILE_NAME_3
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_FILE_NAME_4
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_FILE_NAME_5
 - ProjectGamedev.IO.Tests.TestConstants, [106](#)
- TEST_FILE_NAME_6
 - ProjectGamedev.IO.Tests.TestConstants, [107](#)
- TEST_FILE_NAME_7
 - ProjectGamedev.IO.Tests.TestConstants, [107](#)
- TEST_FILE_NAME_8
 - ProjectGamedev.IO.Tests.TestConstants, [107](#)
- TEST_FILE_NAME_9
 - ProjectGamedev.IO.Tests.TestConstants, [107](#)
- TEST_ROOT_FOLDER
 - ProjectGamedev.IO.Tests.TestConstants, [107](#)
- TEST_THREAD_COUNT
 - ProjectGamedev.IO.Tests.TestConstants, [107](#)
- TestConcurrentAccessToFile
 - ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [13](#)

TestCriticalSectionParameterChecks
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [13](#)
 ProjectGamedev.IO.Tests.TestConstants, [107](#)
 TIMEOUT_MILLISECONDS
 ProjectGamedev.IO.Tests.TestConstants, [107](#)

TestDirectoryMethodsNoTimeoutsNoExceptions
 ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, [66](#)
 WriteAllBytes
 ProjectGamedev.IO.File, [96](#)

testdirectorypath1
 WriteAllLines
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [14](#)
 ProjectGamedev.IO.File, [97–99](#)
 WriteAllText
 ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, [67](#)
 ProjectGamedev.IO.File, [100, 101](#)
 ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)

testdirectorypath2
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [14](#)
 ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, [67](#)
 ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)

TestDualKeyLocking
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [13](#)

TestExclusiveLocking
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [14](#)

TestExclusiveLockingNoTimeout
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [14](#)

TestExclusiveSectionParameterChecks
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [14](#)

TestFileMethodsNoTimeoutsNoExceptions
 ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)

testfilepath1
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [14](#)
 ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, [67](#)
 ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)

testfilepath2
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [14](#)
 ProjectGamedev.IO.Tests.PlayModeTests.DirectoryTest, [67](#)
 ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)

testfilepath3
 ProjectGamedev.IO.Tests.PlayModeTests.ConcurrentIOTest, [15](#)
 ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)

testfilepath4
 ProjectGamedev.IO.Tests.PlayModeTests.FileTest, [103](#)

TIMEOUT_LARGE_MILLISECONDS