# System.IO.Path Class

[ILAsm]
```
.class public sealed Path extends System.Object
```

[C#]
```
public sealed class Path
```

### Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
  - CLSCompliantAttribute(true)

### Summary

Performs operations on `System.String` instances that contain file or directory path information.

### Inherits From: System.Object

**Library:** BCL

**Thread Safety:** All public static members of this type are safe for multithreaded operations. No instance members are guaranteed to be thread safe.

### Description

A path is a string that provides the location of a file or directory. A path does not necessarily point to a location on disk; for example, a path might map to a location in memory or on a device. Paths are composed of the components described below. Component names are shown in *italics* and the following table describes the symbols used in component definitions:

| Symbol | Description |
|---|---|
| < > | Indicates a path component. |
| { } | Indicates a grouping; either all components in a grouping are present, or none are permitted to be present. |
| * | Indicates that the component or grouping that immediately precedes this symbol can appear zero, one, or multiple times. |
| ? | Indicates that the component or grouping that immediately precedes this symbol can appear zero, or one times. |
| + | Indicates string concatenation. |

The components that define a path are as follows:

*Directory Name*: A string that specifies one or more directory levels in a file system. If a directory name contains multiple levels, a *directory separator character* separates the levels; however, a directory name does not begin or end with a directory separator character. In the example path `C:/foo/bar/bat.txt`, the directory name is "`foo/bar`". `System.IO.Path.GetDirectoryName` returns the directory name component of a path. Note that this method does include a beginning separator character if one is included in the specified path.

*Directory Separator Character*: An implementation-specific constant string containing a single printable non-alphanumeric character used to separate levels in a file system. In the example path `C:/foo/bar/bat.txt`, the directory separator character is "`/`". The `System.IO.Path.DirectorySeparatorChar` and `System.IO.Path.AltDirectorySeparatorChar` store implementation-specific directory separator characters.

*Extension*: A string that consists of the characters at the end of a file name, from and including the last *extension separator character*. The minimum and maximum lengths of extension components are implementation-specific. In the example path `C:/foo/bar/bat.txt`, the *extension* is "`.txt`". The `System.IO.Path.GetExtension` method returns the extension component of a path.

*Extension Separator Character*: An implementation-specific constant string composed of a single character that appears after the last character in the *file base* component indicating the beginning of the *extension* component. If the extension separator character is the first character in a *file name*, it is not interpreted as a extension separator character. If more than one extension separator character appears in a file name, only the last occurrence is the extension separator character; all other occurrences are part of the file base component. In the example path `C:/foo/bar/bat.txt`, the extension separator character is "`.`".

*File Base*: A string containing the *filename* with the *extension* component removed. In the example path `C:/foo/bar/bat.txt`, the file base is "bat". The `System.IO.Path.GetFileNameWithoutExtension` method returns the file base component of a path.

*File Name*: A string containing all information required to uniquely identify a file within a directory. This component is defined as follows:

```
<file base>{+<extension>}?
```

The file name component is commonly referred to as a relative file name. In the example path `C:/foo/bar/bat.txt`, the file name is "bat.txt". The `System.IO.Path.GetFileName` method returns the file name component of a path.

*Full Directory Name*: A string containing all information required to uniquely identify a directory within a file system. This component is defined as follows:

`<path root>+<directory name>`

The full directory name component is commonly referred to as the absolute directory name. In the example path `C:/foo/bar/bat.txt`, the full directory name is "`C:/foo/bar` ".

*FullPath*: A string containing all information required to uniquely identify a file within a file system. This component is defined as follows:

`<full directory name>+<directory separator character>+<file name>`

The full path component is commonly referred to as the absolute file name. In the example path `C:/foo/bar/bat.txt`, the full path is "`C:/foo/bar/bat.txt`". The `System.IO.Path.GetFullPath` method returns the full path component.

*Path Root*: A string containing all information required to uniquely identify the highest level in a file system. The component is defined as follows:

```
{<volume identifier>+<volume separator character>}?+<directory
separator character>
```

In the example path `C:/foo/bar/bat.txt`, the path root is "`C:/`". The `System.IO.Path.GetPathRoot` method returns the *path root* component.

*VolumeIdentifier*: A string composed of a single alphabetic character that uniquely defines a drive or volume in a file system. This component is optional; on systems that do not support volume identifiers, this component is required to be a zero length string. In the example path `C:/foo/bar/bat.txt`, the path root is "`C:`". In the example path, `\\myserver\myshare\foo\bar\baz.txt` the path root is "`\\myserver\myshare`".

*Volume Separator Character*: A string composed of a single alphabetic character used to separate the *volumeidentifier* from other components in a path. This component can appear in a path only if a volume identifier is present. This component is optional; on systems that do not support the volume identifier component, the volume separator character component is required to be a zero length string.

The exact format of a path is determined by the current platform. For example, on Windows systems a path can start with a volume identifier, while this element is not present in Unix system paths. On some systems, paths containing file names can contain extensions. The format of an extension is platform dependent; for example, some systems limit extensions to three characters, while others do not. The current platform and possibly the current file system determine the set of characters used to separate the elements of a path, and the set of characters that cannot be used when specifying paths. Because of these differences, the fields of the `System.IO.Path` class as well as the exact behavior of some members of the `System.IO.Path` class are determined by the current platform and/or file system.

A path contains either absolute or relative location information. Absolute paths fully specify a location: the file or directory can be uniquely identified regardless of the current location. A full path or full directory name component is present in an absolute path. Relative paths specify a partial location: the current working directory is used as the starting point when locating a file specified with a relative path. [*Note:* To determine the current working directory, call `System.IO.Directory.GetCurrentDirectory`.]

Most members of the `Path` class do not interact with the file system and do not verify the existence of the file or directory specified by a path string. `System.IO.Path` members that modify a path string, such as `System.IO.Path.ChangeExtension`, have no effect on files and directories in the file system. `System.IO.Path` members do, however, validate the contents of a specified path string, and throw `System.ArgumentException` if the string contains characters that are not valid in path strings, as defined by the current platform and file system. Implementations are required to preserve the case of file and directory path strings, and to be case sensitive if and only if the current platform is case-sensitive.

# Path.AltDirectorySeparatorChar Field

[ILAsm]

```
.field public static initOnly valuetype System.Char
AltDirectorySeparatorChar
```

[C#]

```
public static readonly char AltDirectorySeparatorChar
```

## Summary

Provides a string containing an alternate single printable non-alphanumeric character used to separate directory levels in a hierarchical file system.

## Description

This field is read-only.

This field can be set to the same value as `System.IO.Path.DirectorySeparatorChar`.

[*Note:* `System.IO.Path.AltDirectorySeparatorChar` and `System.IO.Path.DirectorySeparatorChar` are both valid for separating directory levels in a path string.

The value of this field is a slash ('/') on Windows systems and a backslash ('\') on Unix systems.

]

# Path.DirectorySeparatorChar Field

[ILAsm]
```
.field public static initOnly valuetype System.Char
DirectorySeparatorChar
```

[C#]
```
public static readonly char DirectorySeparatorChar
```

### Summary

Provides a string containing a single printable non-alphanumeric character used to separate directory levels in a hierarchical file system.

### Description

This field is read-only.

[*Note:* `System.IO.Path.AltDirectorySeparatorChar` and `System.IO.Path.DirectorySeparatorChar` are both valid for separating directory levels in a path string.

The value of this field is a backslash ('\') on Windows systems and a slash ('/') on Unix systems.

]

# Path.PathSeparator Field

[ILAsm]

```
.field public static initOnly valuetype System.Char PathSeparator
```

[C#]

```
public static readonly char PathSeparator
```

## Summary

Provides a implementation-specific separator character used to separate path strings in environment variables.

## Description

This field is read-only.

# Path.ChangeExtension(System.String, System.String) Method

```
[ILAsm]
.method public hidebysig static string ChangeExtension(string path,
string extension)

[C#]
public static string ChangeExtension(string path, string extension)
```

### Summary

Changes the extension component of the specified path string.

### Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the path information to modify. |
| *extension* | A `System.String` containing the new extension. Specify `null` to remove an existing extension from *path*. |

### Return Value

A `System.String` containing the modified path information.

Platforms that do not support this feature return *path* unmodified.

### Description

The exact behavior of this method is implementation-specific. This method checks *path* for invalid characters as defined by the current platform and file system.

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters. |

# Path.Combine(System.String, System.String) Method

```
[ILAsm]
.method public hidebysig static string Combine(string path1, string
path2)

[C#]
public static string Combine(string path1, string path2)
```

### Summary

Concatenates two path strings.

### Parameters

| Parameter | Description |
|---|---|
| *path1* | A `System.String` containing the first path. |
| *path2* | A `System.String` containing the second path. |

### Return Value

A `System.String` containing *path1* followed by *path2*. If one of the specified paths is a zero length string, this method returns the other path. If *path2* contains an absolute path, this method returns *path2*.

### Description

If *path1* does not end with a valid separator character (`System.IO.Path.DirectorySeparatorChar` or `System.IO.Path.AltDirectorySeparatorChar`), `DirectorySeparatorChar` is appended to *path1* prior to the concatenation.

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentNullException** | *path1* or *path2* is `null`. |
| **System.ArgumentException** | *path1* or *path2* contains one or more implementation-specific invalid characters. |

### Example

The following example demonstrates using the `Combine` method on a Windows system.

[C#]

```
using System;
using System.IO;
class CombineTest {
 public static void Main() {
 string path1, path2;
 Console.WriteLine("Dir char is {0} Alt dir char is {1}",
 Path.DirectorySeparatorChar,
 Path.AltDirectorySeparatorChar
 );
 path1 = "foo.txt";
 path2 = "\\ecmatest\\examples";
 Console.WriteLine("{0} combined with {1} = {2}",path1, path2,
Path.Combine(path1,

path2));
 path1 = "\\ecmatest\\examples";
 path2 = "foo.txt";
 Console.WriteLine("{0} combined with {1} = {2}",path1, path2,
Path.Combine(path1,

path2));
 }
}
```

The output is

```
Dir char is \ Alt dir char is /


foo.txt combined with \ecmatest\examples = \ecmatest\examples


\ecmatest\examples combined with foo.txt = \ecmatest\examples\foo.txt
```

# Path.GetDirectoryName(System.String) Method

```
[ILAsm]
.method public hidebysig static string GetDirectoryName(string path)

[C#]
public static string GetDirectoryName(string path)
```

### Summary

Returns the directory name component of the specified path string.

### Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the path of a file or directory. |

### Return Value

A `System.String` containing directory information for *path*, or `null` if *path* denotes a root directory, is the empty string, or is `null`. Returns `System.String.Empty` if path does not contain directory information.

### Description

The string returned by this method consists of all characters between the first and last `System.IO.Path.DirectorySeparatorChar` or `System.IO.Path.AltDirectorySeparatorChar` character in *path*. The first separator character is included, but the last separator character is not included in the returned string.

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters. |

### Example

The following example demonstrates using the `System.IO.Path.GetDirectoryName` method on a Windows system.

[C#]

```
using System;
using System.IO;
class GetDirectoryTest {
 public static void Main() {
   string [] paths = {
     @"\ecmatest\examples\pathtests.txt",
     @"\ecmatest\examples\",
     "pathtests.xyzzy",
     @"\",
     @"C:\",
    @"\\myserver\myshare\foo\bar\baz.txt"
   };
   foreach (string pathString in paths) {
     string s = Path.GetDirectoryName(pathString);
     Console.WriteLine("Path: {0} directory is {1}",pathString, s==
null? "null": s);
   }
 }
}
```

The output is

```
Path: \ecmatest\examples\pathtests.txt directory is \ecmatest\examples


Path: \ecmatest\examples\ directory is \ecmatest\examples


Path: pathtests.xyzzy directory is


Path: \ directory is null


Path: C:\ directory is null


Path: \\myserver\myshare\foo\bar\baz.txt directory is
\\myserver\myshare\foo\bar
```

# Path.GetExtension(System.String) Method

```
[ILAsm]
.method public hidebysig static string GetExtension(string path)

[C#]
public static string GetExtension(string path)
```

### Summary

Returns the extension component of the specified path string.

### Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the path information from which to get the extension. |

### Return Value

A `System.String` containing the extension of *path*,`null`, or `System.String.Empty`. If *path* is `null`, returns `null`. If path does not have extension information, returns `System.String.Empty`.

The extension returned by this method includes the implementation-specific extension separator character used to separate the extension from the rest of the path.

Platforms that do not support this feature return *path* unmodified.

### Description

The exact behavior of this method is implementation-specific. The character used to separate the extension from the rest of the path is implementation-specific.

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters. |

### Example

The following example demonstrates using the `System.IO.Path.GetExtension` method on a Windows system.

[C#]

```
using System;
using System.IO;
class GetDirectoryTest {
 public static void Main(){
   string [] paths = {
     @"\ecmatest\examples\pathtests.txt",
     @"\ecmatest\examples\",
     "pathtests.xyzzy",
     "pathtests.xyzzy.txt",
     @"\",
     ""
   };
   foreach (string pathString in paths){
     string s = Path.GetExtension (pathString);
     if (s == String.Empty) s= "(empty string)";
     if (s == null) s= "null";
     Console.WriteLine("{0} is the extension of {1}", s, pathString);
   }
 }
}
```

The output is

```
.txt is the extension of \ecmatest\examples\pathtests.txt


(empty string) is the extension of \ecmatest\examples\


.xyzzy is the extension of pathtests.xyzzy


.txt is the extension of pathtests.xyzzy.txt


(empty string) is the extension of \


(empty string) is the extension of
```

# Path.GetFileName(System.String) Method

[ILAsm]

```
.method public hidebysig static string GetFileName(string path)
```

[C#]

```
public static string GetFileName(string path)
```

### Summary

Returns the file name, including the extension if any, of the specified path string.

### Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the path information from which to obtain the filename and extension. |

### Return Value

A `System.String` consisting of the characters after the last directory character in *path*. If the last character of *path* is a directory separator character, returns `System.String.Empty`. If *path* is `null`, returns `null`.

Platforms that do not support this feature return *path* unmodified.

### Description

The directory separator characters used to determine the start of the file name are `System.IO.Path.DirectorySeparatorChar` and `System.IO.Path.AltDirectorySeparatorChar`.

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters. |

### Example

The following example demonstrates the behavior of the `System.IO.Path.GetFileName` method on a Windows system.

[C#]

```
using System;
using System.IO;
class FileNameTest {
 public static void Main() {
   string [] paths = {"pathtests.txt",
     @"\ecmatest\examples\pathtests.txt",
     "c:pathtests.txt",
     @"\ecmatest\examples\",
     ""
   };
   foreach (string p in paths) {
     Console.WriteLine("Path: {0} filename = {1}",p,
Path.GetFileName(p));
   }
 }
}
```

The output is

```
Path: pathtests.txt filename = pathtests.txt


Path: \ecmatest\examples\pathtests.txt filename = pathtests.txt


Path: c:pathtests.txt filename = pathtests.txt


Path: \ecmatest\examples\ filename =


Path: filename =
```

# Path.GetFileNameWithoutExtension(System.String) Method

```
[ILAsm]
.method public hidebysig static string
GetFileNameWithoutExtension(string path)

[C#]
public static string GetFileNameWithoutExtension(string path)
```

### Summary

Returns the file base component of the specified path string without the extension.

### Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the path of the file. |

### Return Value

A `System.String` consisting of the string returned by `System.IO.Path.GetFileName`, minus the implementation-specific extension separator character and extension. Platforms that do not support this feature return *path* unmodified.

### Description

[*Note:* For additional details, see `System.IO.Path.GetFileName`.]

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters. |

# Path.GetFullPath(System.String) Method

```
[ILAsm]
.method public hidebysig static string GetFullPath(string path)

[C#]
public static string GetFullPath(string path)
```

### Summary

Returns information required to uniquely identify a file within a file system.

### Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the file or directory for which to obtain absolute path information. |

### Return Value

A `System.String` containing the fully qualified (absolute) location of *path*.

### Description

The absolute path includes all information required to locate a file or directory on a system. The file or directory specified by *path* is not required to exist; however if *path* does exist, the caller is required to have permission to obtain path information for *path*. Note that unlike most members of the `System.IO.Path` class, this method accesses the file system.

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* is a zero-length string, contains only white space, or contains one or more implementation-specific invalid characters.<br>-or-<br>The system could not retrieve the absolute path. |
| **System.Security.SecurityException** | The caller does not have the required permissions. |

| | |
|---|---|
| **System.ArgumentNullException** | *path* is `null`. |
| **System.IO.PathTooLongException** | The length of *path* or the absolute path information for *path* exceeds the system-defined maximum length. |

### Example

The following example demonstrates the `System.IO.Path.GetFullPath` method on a Windows system. In this example, the absolute path for the current directory is c:\ecmatest\examples.

[C#]

```
using System;
using System.IO;
class GetDirectoryTest {
 public static void Main() {
   string [] paths = {
     @"\ecmatest\examples\pathtests.txt",
     @"\ecmatest\examples\",
     "pathtests.xyzzy",
     @"\",
   };
   foreach (string pathString in paths)
     Console.WriteLine("Path: {0} full path is {1}",pathString,

Path.GetFullPath(pathString));
 }
}
```

The output is

```
Path: \ecmatest\examples\pathtests.txt full path is
C:\ecmatest\examples\pathtests.txt


Path: \ecmatest\examples\ full path is C:\ecmatest\examples\


Path: pathtests.xyzzy full path is C:\ecmatest\examples\pathtests.xyzzy


Path: \ full path is C:\
```

### Permissions

| Permission | Description |
|---|---|
| **System.Security.Permissions.FileIOPermission** | Requires permission to access path information. See `System.Security.Permissions.FileIOPermissionAccess.Path` |

# Path.GetPathRoot(System.String) Method

[ILAsm]
```
.method public hidebysig static string GetPathRoot(string path)
```

[C#]
```
public static string GetPathRoot(string path)
```

## Summary

Returns the path root component of the specified path.

## Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the path from which to obtain root directory information |

## Return Value

A `System.String` containing the root directory of *path*, or `null` if *path* is `null`. Returns `System.String.Empty` if the specified path does not contain root information.

Platforms that do not support this feature return *path* unmodified.

## Description

This method does not verify that the path exists.

The exact behavior of this method is implementation-specific.

## Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters or is equal to `System.String.Empty`. |

## Example

The following example demonstrates the `System.IO.Path.GetPathRoot` method.

[C#]

```
using System;
using System.IO;
class GetPathRootTest
{
  public static void Main() {
    string [] paths = {

@"\ecmatest\examples\pathtests.txt",
      "pathtests.xyzzy",
      @"\",
      @"C:\",

@"\\myserver\myshare\foo\bar\baz.txt"
    };
    foreach (string pathString in paths) {
      string s = Path.GetPathRoot(pathString);
      Console.WriteLine("Path: {0} Path root is {1}",pathString, s==
null? "null": s);
    }
  }
}
```

The output is

```
Path: \ecmatest\examples\pathtests.txt Path root is \


Path: pathtests.xyzzy Path root is


Path: \ Path root is \


Path: C:\ Path root is C:\


Path: \\myserver\myshare\foo\bar\baz.txt Path root is
\\myserver\myshare
```

# Path.GetTempFileName() Method

[ILAsm]
```
.method public hidebysig static string GetTempFileName()
```

[C#]
```
public static string GetTempFileName()
```

## Summary

Returns a unique temporary file name and creates a 0-byte file by that name on disk.

## Return Value

A `System.String` containing the name of the temporary file.

Platforms that do not support this feature return `System.String.Empty`.

# Path.GetTempPath() Method

[ILAsm]
```
.method public hidebysig static string GetTempPath()
```

[C#]
```
public static string GetTempPath()
```

### Summary

Returns the path information of a temporary directory.

### Return Value

A `System.String` containing the full directory name of a temporary directory.

The information returned by this method is implementation-specific. Platforms that do not support this feature return `System.String.Empty`.

### Description

On platforms that provide a mechanism for users to discover this information, (for example by checking an environment variable), implementations of the CLI return the same information as the implementation-specific mechanism.

### Exceptions

| Exception | Condition |
|---|---|
| **System.Security.SecurityException** | The caller does not have the required permission. |

### Permissions

| Permission | Description |
|---|---|
| **System.Security.Permissions.EnvironmentPermission** | Requires unrestricted access to environment variables. See `System.Security.Permissions.PermissionState.Unrestricted`. |

# Path.HasExtension(System.String) Method

```
[ILAsm]
.method public hidebysig static bool HasExtension(string path)

[C#]
public static bool HasExtension(string path)
```

### Summary

Returns a `System.Boolean` indicating whether the specified path includes an extension component.

### Parameters

| Parameter | Description |
|---|---|
| *path* | A `System.String` containing the path to search for an extension. |

### Return Value

`true` if *path* includes a file extension.

Platforms that do not support this feature return `false`.

### Exceptions

| Exception | Condition |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters. |

# Path.IsPathRooted(System.String) Method

```
[ILAsm]
.method public hidebysig static bool IsPathRooted(string path)

[C#]
public static bool IsPathRooted(string path)
```

## Summary

Returns a `System.Boolean` indicating whether the specified path string contains a path root component.

## Parameters

| **Parameter** | **Description** |
|---|---|
| *path* | A `System.String` containing the path to test. |

## Return Value

`true` if *path* contains an absolute path; `false` if *path* contains relative path information.

Platforms that do not support this feature return `false`.

## Description

[*Note:* This method does not access file systems or verify the existence of the specified path.]

## Exceptions

| **Exception** | **Condition** |
|---|---|
| **System.ArgumentException** | *path* contains one or more implementation-specific invalid characters. |