System.Environment Class

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

[C#]
public sealed class Environment
```

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

Provides the current settings for, and information about, the execution environment.

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

[*Note:* Use this class to retrieve the following information:

- Command line arguments
- Exit codes
- Environment variable settings
- Contents of the call stack
- Time since last system boot
- Version of the execution engine

]

Environment.Exit(System.Int32) Method

[ILAsm]
```
.method public hidebysig static void Exit(int32 exitCode)
```

[C#]
```
public static void Exit(int exitCode)
```

Summary

Terminates the current process and sets the process exit code to the specified value.

Parameters

Parameter	Description
exitCode	A `System.Int32` value that is provided to the operating system.

Description

This method causes an executing program to halt.

Exceptions

Exception	Condition
System.Security.SecurityException	The immediate caller does not have the required permission.

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	Requires unmanaged code permission. See `System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode`.

Environment.GetCommandLineArgs() Method

[ILAsm]

```
.method public hidebysig static string[] GetCommandLineArgs()
```

[C#]

```
public static string[] GetCommandLineArgs()
```

Summary

Returns the arguments specified on the command line.

Return Value

Returns a `System.String` array. Each `System.String` in the array contains a single command line argument.

Description

The first element in the array contains the filename of the executing program. If the filename is not available, the first element is equal to `System.String.Empty`. The remaining elements contain any additional tokens entered on the command line.

[*Note:* The program filename can, but is not required to, include path information.

To obtain the command line as a single `System.String`, use the `System.Environment.CommandLine` property.

]

Environment.GetEnvironmentVariable(System.String) Method

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

[C#]
```
public static string GetEnvironmentVariable(string variable)
```

Summary

Returns the value of the specified environment variable.

Parameters

Parameter	Description
variable	A `System.String` containing the name of an environment variable.

Return Value

A `System.String` containing the current setting of *variable*, or `null`.

Description

If *variable* contains a valid name for an environment variable, and if the caller has sufficient permissions, this method returns the current setting for *variable*. Environment variable names are case-insensitive.

If *variable* specifies an invalid name or the system does not support environment variables, this method returns `null`.

[*Note:* To obtain names and settings for all environment variables, use the `System.Environment.GetEnvironmentVariables` method.]

Exceptions

Exception	Condition
System.ArgumentNullException	*variable* is a null reference.
System.Security.SecurityException	The caller does not have the required permission.

Permissions

Permission	Description
System.Security.Permissions.EnvironmentPermission	Requires permission to read environment variables. See `System.Security.Permissions.EnvironmentPermissionAccess.Read`.

Environment.GetEnvironmentVariables() Method

```
[ILAsm]
.method public hidebysig static class System.Collections.IDictionary
GetEnvironmentVariables()

[C#]
public static IDictionary GetEnvironmentVariables()
```

Summary

Returns all environment variables and their current settings.

Return Value

A `System.Collections.IDictionary` object containing environment variable names and settings, or `null` if the system does not support environment variables.

Description

The names and settings for the environment variables are stored in the returned `System.Collections.IDictionary` object as keys and values, respectively.

[*Note:* To obtain the setting of a single environment variable, use the `System.Environment.GetEnvironmentVariable` method.]

Exceptions

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

Example

The following example prints the names and values of all environment variables defined in the environment.

[C#]

```
using System;
using System.Collections;
```

```
class EnvTest:Object {
  public static void Main() {
    Console.WriteLine("Environment Variables");
    IDictionary envars =
        Environment.GetEnvironmentVariables();
    IDictionaryEnumerator varEnumerator =
        envars.GetEnumerator();
    while(varEnumerator.MoveNext() != false) {
      Console.WriteLine("{0}={1}",
                        varEnumerator.Key,
                        varEnumerator.Value);
    }
  }
}
```

The output will vary depending on your system.

Permissions

Permission	Description
System.Security.Permissions. EnvironmentPermission	Requires permission to read environment variables. See `System.Security.Permissions.EnvironmentPermissionAccess.Read`.

Environment.CommandLine Property

[ILAsm]

```
.property string CommandLine { public hidebysig static specialname
string get_CommandLine() }
```

[C#]

```
public static string CommandLine { get; }
```

Summary

Gets the information entered on the command line when the current process was started.

Property Value

A `System.String` containing the command line arguments.

Description

This property is read-only.

This property provides access to the program name and any arguments specified on the command line when the current process was started.

If the environment does not support a program name, as can be the case with compact devices, then the program name is equal to `System.String.Empty`.

The format of the information returned by this property is implementation-specific.

[*Note:* The program name can, but is not required to, include path information.

Use the `System.Environment.GetCommandLineArgs` method to retrieve the command line information parsed and stored in an array of strings.

]

Environment.ExitCode Property

[ILAsm]

```
.property int32 ExitCode { public hidebysig static specialname int32
get_ExitCode() public hidebysig static specialname void
set_ExitCode(int32 value) }
```

[C#]

```
public static int ExitCode { get; set; }
```

Summary

Gets or sets the exit code of a process.

Property Value

A `System.Int32` value returned by a process. The default value is zero.

Description

When a process exits, if the process does not return a value, the value of `System.Environment.ExitCode` is returned. If the value of this property is not set by an application, zero is returned.

On operating systems that do not support process exit codes, CLI implementations are required to fully support getting and setting values for this property.

Environment.HasShutdownStarted Property

[ILAsm]

```
.property bool HasShutdownStarted { public hidebysig static
specialname instance bool get_HasShutdownStarted() }
```

[C#]

```
public static bool HasShutdownStarted { get; }
```

Summary

Gets a value indicating whether an application has started to shut down.

Property Value

A `System.Boolean` where `true` indicates the shutdown process has started; otherwise `false`.

Description

This property is read-only.

[*Note:* This property is for use inside the finalizer of an application. If the shutdown process has started, static members should not be accessed; they might have been cleaned up by the garbage collector. If the member has been cleaned up, any access attempt will cause an exception to be thrown.

`System.Console.Out` is a special case that is always available after the shutdown process has started.

]

Environment.NewLine Property

[ILAsm]
```
.property string NewLine { public hidebysig static specialname
string get_NewLine() }
```

[C#]
```
public static string NewLine { get; }
```

Summary

Gets the newline string for the current platform.

Property Value

A `System.String` containing the characters required to write a newline.

Description

This property is read-only.

[*Note:* This property is intended for platform-independent formatting of multi-line strings. This value is automatically appended to text when using `WriteLine` methods, such as `System.Console.WriteLine`.]

Example

The following example demonstrates using the `System.Environment.NewLine` property. The string returned by `System.Environment.NewLine` is inserted between "Hello" and "World", causing a line break between the words in the output.

[C#]

```
using System;
class TestClass {
    public static void Main() {
      Console.WriteLine("Hello,{0}World",
                        Environment.NewLine);
    }
}
```

The output is

```
Hello,


World
```

Environment.StackTrace Property

[ILAsm]
```
.property string StackTrace { public hidebysig static specialname
string get_StackTrace() }
```

[C#]
```
public static string StackTrace { get; }
```

Summary

Returns a string representation of the state of the call stack.

Property Value

A `System.String` containing a description of the methods currently in the call stack. This value can be `System.String.Empty`.

Description

This property is read-only.

[*Note:* An example of how the `System.String` returned by this property might be formatted follows, where one line of information is provided for each method on the call stack:

`at` *FullClassName*`.`*MethodName*`(`*MethodParms*`) in` *FileName*`:line` *LineNumber*

FullClassName, *MethodName*, *MethodParms*, *FileName*, and *LineNumber* are defined as follows:

Item	Description
FullClassName	The fully qualified name of the class.
MethodName	The name of the method.
MethodParms	The list of parameter type/name pairs. Each pair is separated by a comma (,). This information is omitted if *MethodName* takes zero parameters.
FileName	The name of the source file where the *MethodName* method is declared. This information is omitted if debug symbols are not available.
LineNumber	The number of the line in *FileName* that contains the source code from *MethodName* for the instruction that is on the call stack. This information is omitted if debug symbols are not available.

The literal "at" is preceded by a single space.

The literals "in" and ":line" are omitted if debug symbols are not available.

The method calls are described in reverse chronological order (the most recent method call is described first).

`System.Environment.StackTrace` might not report as many method calls as expected, due to code transformations that occur during optimization.

]

Example

The following example gets the `System.Environment.StackTrace` property from within a series of nested calls.

[C#]

```
using System;
public class TestCallStack {
    public void MyMethod1 () {
        MyMethod2();
    }
    public void MyMethod2 () {
        MyMethod3();
    }
    public void MyMethod3 () {
        Console.WriteLine("TestCallStack: {0}",
                          Environment.StackTrace);
    }
    public static void Main() {
        TestCallStack t = new TestCallStack();
        t.MyMethod1();
    }
}
```

Without debug symbols the output is

```
TestCallStack: at System.Environment.GetStackTrace(Exception e)


at System.Environment.GetStackTrace(Exception e)


at System.Environment.get_StackTrace()


at TestCallStack.Main()
```

With debug symbols the output is

```
TestCallStack: at System.Environment.GetStackTrace(Exception e)


at System.Environment.GetStackTrace(Exception e)
at System.Environment.get_StackTrace()

at TestCallStack.MyMethod3() in c:\ECMAExamples\envstack.cs:line 10

at TestCallStack.MyMethod2() in c:\ECMAExamples\envstack.cs:line 8

at TestCallStack.MyMethod1() in c:\ECMAExamples\envstack.cs:line 5

at TestCallStack.Main() in c:\ECMAExamples\envstack.cs:line 15
```

Environment.TickCount Property

[ILAsm]
```
.property int32 TickCount { public hidebysig static specialname
int32 get_TickCount() }
```

[C#]
```
public static int TickCount { get; }
```

Summary

Gets the number of milliseconds elapsed since the system was started.

Property Value

A `System.Int32` value containing the amount of time in milliseconds that has passed since the last time the computer was started.

Description

This property is read-only.

The resolution of the `System.Environment.TickCount` property cannot be less than 500 milliseconds.

The value of this property is derived from the system timer.

The `System.Environment.TickCount` property handles an overflow condition by resetting its value to zero. The minimum value returned by `System.Environment.TickCount` is 0.

[*Note:* `System.Environment.TickCount` is measured in milliseconds, not in "ticks".

The `System.Environment.TickCount` reaches its maximum value after approximately 24.8 days of continuous up time.

For applications that require a finer granularity or a larger maximum time than `System.Environment.TickCount` supports, see `System.DateTime.Now`.

]

Environment.Version Property

[ILAsm]

```
.property class System.Version Version { public hidebysig static
specialname class System.Version get_Version() }
```

[C#]

```
public static Version Version { get; }
```

Summary

Gets the current version of the execution engine.

Property Value

A `System.Version` object that contains the major, minor, build, and revision numbers of the execution engine.

Description

This property is read-only.