.NET Framework Class Library

**AppDomain.UnhandledException Event**

Occurs when an exception is not caught.

**Namespace:**  [System](http://msdn.microsoft.com/en-us/library/system.aspx)  
**Assembly:**  mscorlib (in mscorlib.dll)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

Visual Basic

Public Event UnhandledException As UnhandledExceptionEventHandler

C#

public event UnhandledExceptionEventHandler UnhandledException

Visual C++

public:

virtual event UnhandledExceptionEventHandler^ UnhandledException {

void add (UnhandledExceptionEventHandler^ value);

void remove (UnhandledExceptionEventHandler^ value);

}

F#

abstract UnhandledException : IEvent<UnhandledExceptionEventHandler,

UnhandledExceptionEventArgs>

override UnhandledException : IEvent<UnhandledExceptionEventHandler,

UnhandledExceptionEventArgs>

**Implements**

[\_AppDomain..::.UnhandledException](http://msdn.microsoft.com/en-us/library/system._appdomain.unhandledexception.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

This event provides notification of uncaught exceptions. It allows the application to log information about the exception before the system default handler reports the exception to the user and terminates the application. If sufficient information about the state of the application is available, other actions may be undertaken — such as saving program data for later recovery. Caution is advised, because program data can become corrupted when exceptions are not handled.

|  |
| --- |
| **NoteNote** |
| In the .NET Framework versions 1.0 and 1.1, application termination and debugging options are reported to the user before this event is raised, rather than after. |

This event can be handled in any application domain. However, the event is not necessarily raised in the application domain where the exception occurred. An exception is unhandled only if the entire stack for the thread has been unwound without finding an applicable exception handler, so the first place the event can be raised is in the application domain where the thread originated.

|  |
| --- |
| **NoteNote** |
| In the .NET Framework versions 1.0 and 1.1, this event occurs only for the default application domain that is created by the system when an application is started. If an application creates additional application domains, specifying a delegate for this event in those applications domains has no effect. |

If the **UnhandledException** event is handled in the default application domain, it is raised there for any unhandled exception in any thread, no matter what application domain the thread started in. If the thread started in an application domain that has an event handler for **UnhandledException**, the event is raised in that application domain. If that application domain is not the default application domain, and there is also an event handler in the default application domain, the event is raised in both application domains.

For example, suppose a thread starts in application domain "AD1", calls a method in application domain "AD2", and from there calls a method in application domain "AD3", where it throws an exception. The first application domain in which the **UnhandledException** event can be raised is "AD1". If that application domain is not the default application domain, the event can also be raised in the default application domain.

|  |
| --- |
| **NoteNote** |
| The common language runtime suspends thread aborts while event handlers for the **UnhandledException** event are executing. |

If the event handler has a [ReliabilityContractAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.constrainedexecution.reliabilitycontractattribute.aspx) attribute with the appropriate flags, the event handler is treated as a constrained execution region.

Starting with the .NET Framework version 4, this event is not raised for exceptions that corrupt the state of the process, such as stack overflows or access violations, unless the event handler is security-critical and has the [HandleProcessCorruptedStateExceptionsAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.exceptionservices.handleprocesscorruptedstateexceptionsattribute.aspx) attribute.

In the .NET Framework versions 1.0 and 1.1, an unhandled exception that occurs in a thread other than the main application thread is caught by the runtime and therefore does not cause the application to terminate. Thus, it is possible for the **UnhandledException** event to be raised without the application terminating. Starting with the .NET Framework version 2.0, this backstop for unhandled exceptions in child threads was removed, because the cumulative effect of such silent failures included performance degradation, corrupted data, and lockups, all of which were difficult to debug. For more information, including a list of cases in which the runtime does not terminate, see [Exceptions in Managed Threads](http://msdn.microsoft.com/en-us/library/ms228965.aspx).

To register an event handler for this event, you must have the required permissions, or a [SecurityException](http://msdn.microsoft.com/en-us/library/system.security.securityexception.aspx) is thrown.

For more information about handling events, see [Consuming Events](http://msdn.microsoft.com/en-us/library/2ccyd347.aspx).

**Other Events for Unhandled Exceptions**

For certain application models, the **UnhandledException** event can be preempted by other events if the unhandled exception occurs in the main application thread.

In applications that use Windows Forms, unhandled exceptions in the main application thread cause the [Application..::.ThreadException](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.threadexception.aspx) event to be raised. If this event is handled, the default behavior is that the unhandled exception does not terminate the application, although the application is left in an unknown state. In that case, the **UnhandledException** event is not raised. This behavior can be changed by using the application configuration file, or by using the [Application..::.SetUnhandledExceptionMode](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.setunhandledexceptionmode.aspx) method to change the mode to UnhandledExceptionMode..::.ThrowException before the [ThreadException](http://msdn.microsoft.com/en-us/library/system.windows.forms.application.threadexception.aspx) event handler is hooked up. This applies only to the main application thread. The **UnhandledException** event is raised for unhandled exceptions thrown in other threads.

Starting with Microsoft Visual Studio 2005, the Visual Basic application framework provides another event for unhandled exceptions in the main application thread. See the [WindowsFormsApplicationBase..::.UnhandledException](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.applicationservices.windowsformsapplicationbase.unhandledexception.aspx) event. This event has an event arguments object with the same name as the event arguments object used by **AppDomain..::.UnhandledException**, but with different properties. In particular, this event arguments object has an [ExitApplication](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.applicationservices.unhandledexceptioneventargs.exitapplication.aspx) property that allows the application to continue running, ignoring the unhandled exception (and leaving the application in an unknown state). In that case, the **AppDomain..::.UnhandledException** event is not raised.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

The following sample demonstrates the **UnhandledException** event.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl00_code');" \o "Copy Code)

Sub Main()

Dim currentDomain As AppDomain = AppDomain.CurrentDomain

AddHandler currentDomain.UnhandledException, AddressOf MyHandler

Try

Throw New Exception("1")

Catch e As Exception

Console.WriteLine("Catch clause caught : " + e.Message)

End Try

Throw New Exception("2")

' Output:

' Catch clause caught : 1

' MyHandler caught : 2

End Sub 'Main

Sub MyHandler(sender As Object, args As UnhandledExceptionEventArgs)

Dim e As Exception = DirectCast(args.ExceptionObject, Exception)

Console.WriteLine("MyHandler caught : " + e.Message)

End Sub 'MyUnhandledExceptionEventHandler

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.Security.Permissions;

public class Test {

[SecurityPermission(SecurityAction.Demand, Flags=SecurityPermissionFlag.ControlAppDomain)]

public static void Example()

{

AppDomain currentDomain = AppDomain.CurrentDomain;

currentDomain.UnhandledException += new UnhandledExceptionEventHandler(MyHandler);

try {

throw new Exception("1");

} catch (Exception e) {

Console.WriteLine("Catch clause caught : " + e.Message);

}

throw new Exception("2");

// Output:

// Catch clause caught : 1

// MyHandler caught : 2

}

static void MyHandler(object sender, UnhandledExceptionEventArgs args) {

Exception e = (Exception) args.ExceptionObject;

Console.WriteLine("MyHandler caught : " + e.Message);

}

public static void Main() {

Example();

}

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl48_ctl00_ctl02_code');" \o "Copy Code)

public ref class Test

{

private:

static void MyHandler( Object^ /\*sender\*/, UnhandledExceptionEventArgs^ args )

{

Exception^ e = dynamic\_cast<Exception^>(args->ExceptionObject);

Console::WriteLine( "MyHandler caught : {0}", e->Message );

}

public:

[SecurityPermissionAttribute( SecurityAction::Demand, ControlAppDomain = true )]

static void Main()

{

AppDomain^ currentDomain = AppDomain::CurrentDomain;

currentDomain->UnhandledException += gcnew UnhandledExceptionEventHandler( Test::MyHandler );

try

{

throw gcnew Exception( "1" );

}

catch ( Exception^ e )

{

Console::WriteLine( "Catch clause caught : {0}", e->Message );

}

throw gcnew Exception( "2" );

// Output:

// Catch clause caught : 1

// MyHandler caught : 2

}

};

int main()

{

Test::Main();