ASP.NET - MULTI THREADING

A thread is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations such as database access or some intense I/O operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increases efficiency of an application.

So far we compiled programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute multiple tasks at a time, it could be divided into smaller threads.

In .Net, the threading is handled through the 'System.Threading' namespace. Creating a variable of the *System.Threading.Thread* type allows you to create a new thread to start working with. It allows you to create and access individual threads in a program.

Creating Thread

A thread is created by creating a Thread object, giving its constructor a ThreadStart reference.

```
ThreadStart childthreat = new ThreadStart(childthreadcall);
```

Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread:

- **The Unstarted State** : It is the situation when the instance of the thread is created but the Start method is not called.
- **The Ready State** : It is the situation when the thread is ready to execute and waiting CPU cycle.
- **The Not Runnable State** : a thread is not runnable, when:
  - Sleep method has been called
  - Wait method has been called
  - Blocked by I/O operations
- **The Dead State** : It is the situation when the thread has completed execution or has been aborted.

Thread Priority

The Priority property of the Thread class specifies the priority of one thread with respect to other. The .Net runtime selects the ready thread with the highest priority.

The priorities could be categorized as:

- Above normal
- Below normal
- Highest
- Lowest
- Normal

Once a thread is created, its priority is set using the Priority property of the thread class.

```
NewThread.Priority = ThreadPriority.Highest;
```

Thread Properties & Methods

The Thread class has the following important properties:

Property	Description
CurrentContext	Gets the current context in which the thread is executing.
CurrentCulture	Gets or sets the culture for the current thread.
CurrentPrinciple	Gets or sets the thread's current principal for role-based security.
CurrentThread	Gets the currently running thread.
CurrentUICulture	Gets or sets the current culture used by the Resource Manager to look up culture-specific resources at run time.
ExecutionContext	Gets an ExecutionContext object that contains information about the various contexts of the current thread.
IsAlive	Gets a value indicating the execution status of the current thread.
IsBackground	Gets or sets a value indicating whether or not a thread is a background thread.
IsThreadPoolThread	Gets a value indicating whether or not a thread belongs to the managed thread pool.
ManagedThreadId	Gets a unique identifier for the current managed thread.
Name	Gets or sets the name of the thread.
Priority	Gets or sets a value indicating the scheduling priority of a thread.
ThreadState	Gets a value containing the states of the current thread.

The Thread class has the following important methods:

Methods	Description
Abort	Raises a ThreadAbortException in the thread on which it is invoked, to begin the process of terminating the thread. Calling this method usually terminates the thread.
AllocateDataSlot	Allocates an unnamed data slot on all the threads. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
AllocateNamedDataSlot	Allocates a named data slot on all threads. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
BeginCriticalRegion	Notifies a host that execution is about to enter a region of code in which the effects of a thread abort or unhandled exception might endanger other tasks in the application domain.
BeginThreadAffinity	Notifies a host that managed code is about to execute instructions that depend on the identity of the current physical operating

	system thread.
EndCriticalRegion	Notifies a host that execution is about to enter a region of code in which the effects of a thread abort or unhandled exception are limited to the current task.
EndThreadAffinity	Notifies a host that managed code has finished executing instructions that depend on the identity of the current physical operating system thread.
FreeNamedDataSlot	Eliminates the association between a name and a slot, for all threads in the process. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
GetData	Retrieves the value from the specified slot on the current thread, within the current thread's current domain. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
GetDomain	Returns the current domain in which the current thread is running.
GetDomainID	Returns a unique application domain identifier.
GetNamedDataSlot	Looks up a named data slot. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
Interrupt	Interrupts a thread that is in the WaitSleepJoin thread state.
Join	Blocks the calling thread until a thread terminates, while continuing to perform standard COM and SendMessage pumping. This method has different overloaded forms.
MemoryBarrier	Synchronizes memory access as follows: The processor executing the current thread cannot reorder instructions in such a way that memory accesses prior to the call to MemoryBarrier execute after memory accesses that follow the call to MemoryBarrier.
ResetAbort	Cancels an Abort requested for the current thread.
SetData	Sets the data in the specified slot on the currently running thread, for that thread's current domain. For better performance, use fields marked with the ThreadStaticAttribute attribute instead.
Start	Starts a thread.
Sleep	Makes the thread pause for a period of time.
SpinWait	Causes a thread to wait the number of times defined by the iterations parameter.
VolatileRead	Reads the value of a field. The value is the latest written by any processor in a computer, regardless of the number of processors or the state of processor cache. This method has different overloaded forms.
VolatileWrite	Writes a value to a field immediately, so that the value is visible to all processors in the computer. This method has different overloaded forms.
Yield	Causes the calling thread to yield execution to another thread that is ready to run on the current processor. The operating system selects the thread to yield to.

Example

The following example illustrates the uses of the Thread class. The page has a label control for displaying messages from the child thread. The messages from the main program are directly displayed using the Response.Write method. Hence they appear on the top of the page.

The source file is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="threaddemo._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

   <head runat="server">
      <title>
         Untitled Page
      </title>
   </head>

   <body>
      <form >
         <div>
            <h3>Thread Example</h3>
         </div>

         <asp:Label ID="lblmessage" runat="server" Text="Label">
         </asp:Label>
      </form>
   </body>

</html>
```

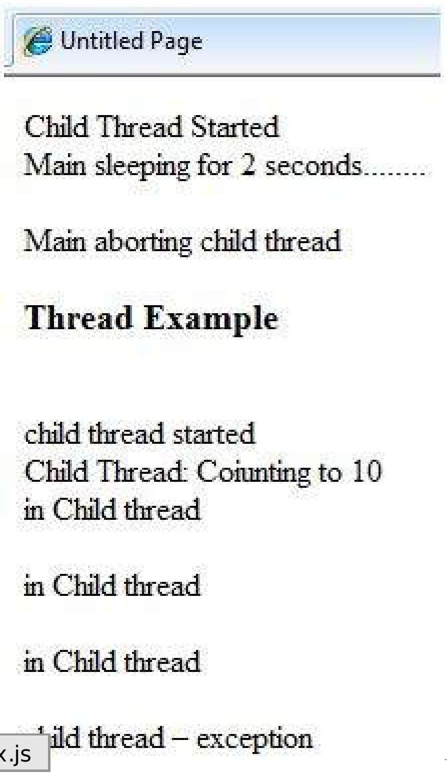
The code behind file is as follows:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;
using System.Threading;

namespace threaddemo
{
   public partial class _Default : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         ThreadStart childthreat = new ThreadStart(childthreadcall);
         Response.Write("Child Thread Started <br/>");
         Thread child = new Thread(childthreat);

         child.Start();

         Response.Write("Main sleeping  for 2 seconds.......<br/>");
         Thread.Sleep(2000);
         Response.Write("<br/>Main aborting child thread<br/>");

         child.Abort();
      }
      public void childthreadcall()
      {
         try{
            lblmessage.Text = "<br />Child thread started <br/>";
            lblmessage.Text += "Child Thread: Coiunting to 10";

            for( int i =0; i<10; i++)
            {
               Thread.Sleep(500);
               lblmessage.Text += "<br/> in Child thread </br>";
            }

            lblmessage.Text += "<br/> child thread finished";

         }catch(ThreadAbortException e){

            lblmessage.Text += "<br /> child thread - exception";

         }finally{
            lblmessage.Text += "<br /> child thread - unable to catch the  exception";
         }
      }
   }
}
```

Observe the following

- When the page is loaded, a new thread is started with the reference of the method childthreadcall. The main thread activities are displayed directly on the web page.
- The second thread runs and sends messages to the label control.
- The main thread sleeps for 2000 ms, during which the child thread executes.
- The child thread runs till it is aborted by the main thread. It raises the ThreadAbortException and is terminated.
- Control returns to the main thread.

When executed the program sends the following messages:



```
Child Thread Started
Main sleeping for 2 seconds........

Main aborting child thread

Thread Example

child thread started
Child Thread: Coiunting to 10
in Child thread

in Child thread

in Child thread

ild thread – exception
```