



## Primitivas de Sincronización Comunes: **ManualResetEventSlim**

Una primitiva de sincronización es un mecanismo por el cual un sistema operativo permite a sus usuarios, en este caso el motor de tiempo de ejecución de .NET, controlar la sincronización de Hilos. La biblioteca *Task Parallel* soporta un amplio rango de primitivas de sincronización que nos permiten controlar el acceso a los recursos de diversas maneras.

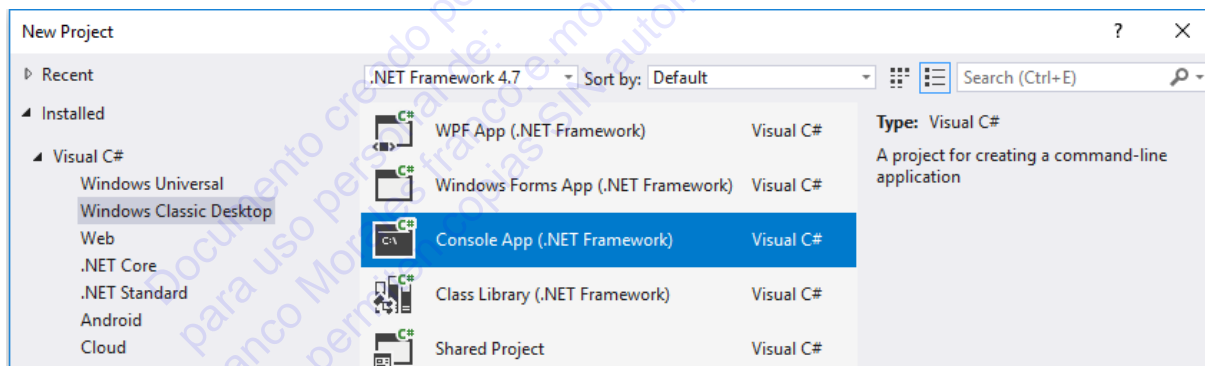
Una de las primitivas de sincronización es la clase **ManualResetEventSlim**. La clase *ManualResetEventSlim* permite que uno o más hilos esperen un evento de señalización. Un objeto *ManualResetEventSlim* se puede encontrar en uno de dos estados: **Signaled** y **NonSignaled**.

### Ejercicio

#### Utilizando **ManualResetEventSlim** con **Task Parallel Library**

En este ejercicio conocerás el uso de la primitiva de sincronización *ManualResetEventSlim* soportada por la biblioteca **Task Parallel**.

1. Crea una aplicación de **Consola** utilizando la plantilla **Console App (.NET Framework)**.



2. Agrega la siguiente instrucción al inicio del archivo **Program.cs** para importar el espacio de nombre de la clase **Thread**.

```
using System.Threading;
```

3. Agrega el siguiente código dentro de la clase **Program** para definir un método que nos permita ejemplificar el uso de la primitiva **ManualResetEventSlim**.

```
static void DoManualResetEventSlim()  
{  
}
```

4. Agrega el siguiente código dentro del método **DoManualResetEventSlim** para crear una instancia de la clase **ManualResetEventSlim**. Su estado predeterminado será **NonSignaled**.

```
ManualResetEventSlim M1 = new ManualResetEventSlim();
```



5. Agrega el siguiente código dentro del método **DoManualResetEventSlim** para crear una instancia de la clase *ManualResetEventSlim* especificando explícitamente su estado como **NonSignaled**.

```
ManualResetEventSlim M2 = new ManualResetEventSlim(false);
```

6. Agrega el siguiente código dentro del método **DoManualResetEventSlim** para crear una instancia de la clase *ManualResetEventSlim* especificando explícitamente su estado como **Signaled**.

```
ManualResetEventSlim M3 = new ManualResetEventSlim(true);
```

7. Agrega el siguiente código dentro del método **DoManualResetEventSlim** para definir una tarea asíncrona.

```
static void DoManualResetEventSlim()  
{  
    ManualResetEventSlim M1 = new ManualResetEventSlim();  
    ManualResetEventSlim M2 = new ManualResetEventSlim(false);  
    ManualResetEventSlim M3 = new ManualResetEventSlim(true);  
  
    var T1 = Task.Run(()=>  
    {  
    });  
}
```

Si un hilo invoca al método **Wait** de un objeto *ManualResetEventSlim* y el objeto se encuentra en estado **NonSignaled**, el hilo es bloqueado hasta que el estado del objeto *ManualResetEventSlim* cambie su estado a **Signaled**.

8. Agrega el siguiente código dentro de la tarea **T1** para que esta espere a que el objeto **M1** cambie su estado a **Signaled** y que envíe a la consola un mensaje cuando esto suceda.

```
var T1 = Task.Run(()=>  
{  
    M1.Wait();  
    Console.WriteLine("M1 está en estado Signaled");  
});
```

En la lógica de la tarea podemos invocar los métodos **Set** o **Reset** del objeto *ManualResetEventSlim* para cambiar su estado a **Signaled** o **NonSignaled** respectivamente.

9. Agrega el siguiente código dentro de la tarea **T1** para cambiar el estado de los objetos **M2** y **M3**.

```
var T1 = Task.Run(()=>  
{  
    M1.Wait();  
    Console.WriteLine("M1 está en estado Signaled");
```



```
Console.WriteLine("Estableciendo estado NonSignaled al objeto M3...");  
M3.Reset();  
Console.WriteLine("Estableciendo estado Signaled al objeto M2...");  
M2.Set();  
});
```

Cuando la tarea *T1* se ejecute, el objeto *M3* se encontrará en estado *Signaled* ya que el objeto *M1* está esperando a ser señalizado y por lo tanto el hilo de la tarea *T1* se encuentra detenido.

10. Agrega el siguiente código después de la declaración de la tarea *T1* para poder mostrar el estado de *M3*.

```
var T1 = Task.Run(()=>  
{  
    M1.Wait();  
    Console.WriteLine("M1 está en estado Signaled");  
    Console.WriteLine("Estableciendo estado NonSignaled al objeto M3...");  
    M3.Reset();  
    Console.WriteLine("Estableciendo estado Signaled al objeto M2...");  
    M2.Set();  
});  
  
Console.WriteLine(  
    $"En el hilo principal ¿Estado de M3 es Signaled? {M3.IsSet}");
```

11. Agrega el siguiente código para señalizar a *M1*. Eso hará que la tarea *T1* continúe su ejecución.

```
Console.WriteLine("El hilo principal señalizará a M1");  
M1.Set();
```

12. Agrega el siguiente código para esperar a que el objeto *M2* sea señalizado. Eso hará que el hilo principal se detenga hasta que *M2* reciba el evento de señalización (al final de la tarea *T1*).

```
Console.WriteLine("El hilo principal espera a que M2 sea señalizado");  
M2.Wait();
```

Después de que *M2* sea señalizado, el hilo principal continuará su ejecución y el estado de *M3* será *NonSignaled* ya que fue restablecido en el código de la tarea *T1*.

13. Agrega el siguiente código para mostrar el estado de *M3*.

```
Console.WriteLine(  
    "En el hilo principal M2 ha sido señalizado");  
  
// Ahora debe mostrar false ya que M3 fue establecido a NonSignaled  
Console.WriteLine(  
    $"En el hilo principal ¿Estado de M3 es Signaled? {M3.IsSet}");
```



Es una buena práctica invocar el método **Dispose** del objeto **ManualResetEventSlim** cuando hayamos terminado de utilizarlo.

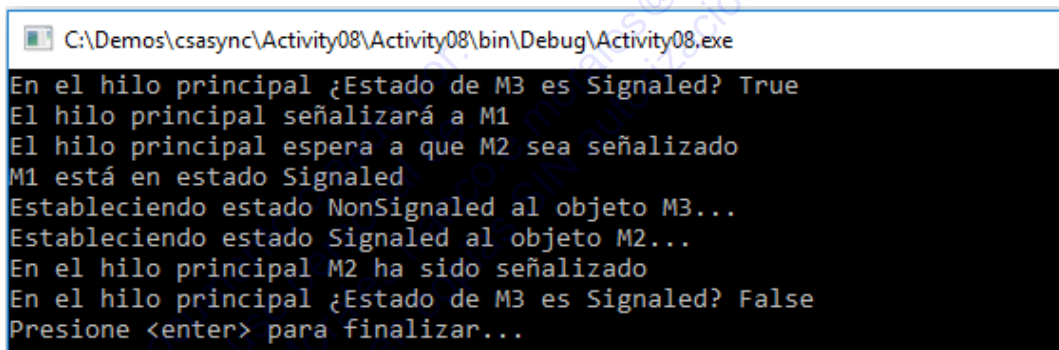
14. Agrega el siguiente código para esperar a que la tarea **T1** haya finalizado completamente antes de hacer **Dispose** a los objetos **ManualResetEventSlim**.

```
T1.Wait();  
M1.Dispose();  
M2.Dispose();  
M3.Dispose();
```

15. Agrega el siguiente código dentro del método **Main** para invocar al método **DoManualResetEventSlim**.

```
DoManualResetEventSlim();  
Console.WriteLine("Presione <enter> para finalizar...");  
Console.ReadLine();
```

16. Ejecuta la aplicación.



```
C:\Demos\csasync\Activity08\Activity08\bin\Debug\Activity08.exe  
En el hilo principal ¿Estado de M3 es Signaled? True  
El hilo principal señalará a M1  
El hilo principal espera a que M2 sea señalado  
M1 está en estado Signaled  
Estableciendo estado NonSignaled al objeto M3...  
Estableciendo estado Signaled al objeto M2...  
En el hilo principal M2 ha sido señalado  
En el hilo principal ¿Estado de M3 es Signaled? False  
Presione <enter> para finalizar...
```

Puedes notar que la ejecución se encuentra sincronizada tal y como lo muestra la secuencia de mensajes.

Normalmente la clase **ManualResetEventSlim** es utilizada para asegurar que sólo un hilo pueda acceder a un recurso en un momento dado.

En este ejercicio mostramos el uso de la primitiva de sincronización **ManualResetEventSlim** soportada por la biblioteca **Task Parallel**.



Para obtener más información sobre la clase **ManualResetEventSlim**, se recomienda consultar el siguiente enlace:

**ManualResetEventSlim Class**

<http://go.microsoft.com/fwlink/?LinkID=267850>