Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

23

Windows Workflow Foundation

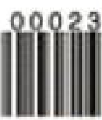
Los componentes y su arquitectura / Introducción a diseñadores de Workflows

Servicios de Runtime

www.reduserspremium.blogspot.com.ar

Scheduling / CommitWorkBatch / Persistence & Tracking services

ISBN 978-987-1347-43-8



00023
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

EL SITIO Nº1 DE TECNOLOGÍA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

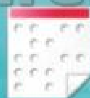
SUSCRIBITE

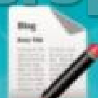
SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

www.reduserspremium.blogspot.com.ar

Foros


Encuestas


Tutoriales

Agenda de eventos

Videos

¡Y mucho más!

redusers.com

Seguinos en:

www.facebook.com/redusers

www.twitter.com/redusers

www.youtube.com/redusersvideos

Servicios de tiempo de ejecución

En las siguientes páginas proporcionaremos una descripción detallada de cada uno de los servicios de WWF.

Anteriormente, en este mismo capítulo, nos referimos a estos servicios que forman parte de la arquitectura de los componentes de Windows Workflow Foundation de forma introductoria.

Scheduling Service

Este servicio es el encargado de la planificación de las tareas para la ejecución de los distintos flujos de trabajo dentro del motor. Estos servicios se pueden implementar de forma asíncrona (que es el servicio predeterminado que nos provee Windows Workflow Foundation), o asíncrona. A estos servicios se los conoce como **DefaultWorkflowShedulerService** y **ManualWorkflowShedulerService**, respectivamente.

Debemos tener en cuenta que el motor de Windows Workflow Foundation no crea hilos de trabajo propios. Los hilos sobre los que se ejecutan las instancias de los flujos de trabajo son hilos de la aplicación que hospeda el motor (aplicación host). De este modo, cada instancia (aunque sea del mismo flujo de trabajo) puede ejecutarse sobre diferentes hilos y WorkflowSchedulerService es el encargado de gestionar estos hilos.

Como ya hemos mencionado, el servicio que se crea por defecto es el **DefaultWorkflowSchedulerService**. Sin embargo, no es necesario especificarlo ya que al iniciar el motor, si no se añade un servicio manual o personalizado, WindowsWorkflowFoundation utilizará éste. Por el contrario, el servicio manual que nos provee WindowsWorkflowFoundation, **ManualWorkflowSchedulerService**, usa una gestión síncrona, de modo que las instancias de los flujos de trabajo serán ejecutadas sobre el mismo hilo de la aplicación que hospeda el motor, y bloqueará así su ejecución hasta que la instancia quede inactiva.

Analicemos el funcionamiento de un **DefaultWorkflowSchedulerService**. Los flujos de trabajo se almacenan en el interior de la cola DefaultWorkflowSchedulerService, a la espera de ser ejecutados. Entonces, cuando el DefaultWorkflowSchedulerService quiere iniciar un flujo de trabajo, se toma un hilo del pool de hilos del .NET Framework y se lo utiliza para ejecutar el flujo de trabajo.

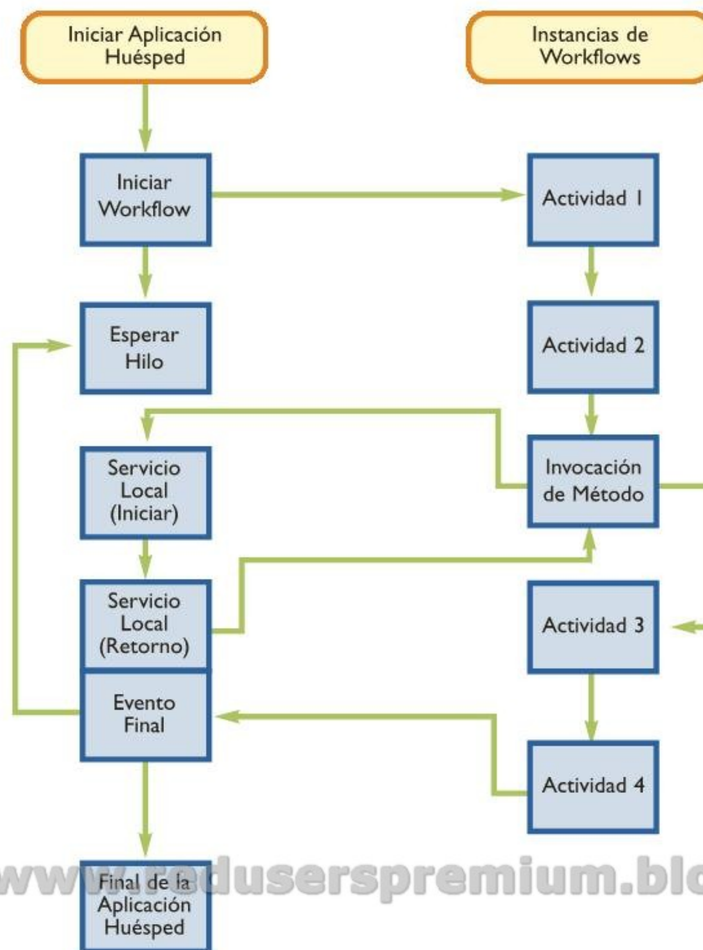
El servicio posee una propiedad llamada **MaxSimultaneousWorkflows**, que deter-

En Windows Workflow Foundation también podemos crear nuestro propio servicio de planificación al heredar la clase WorkflowSchedulerService.

www.reduserspremium.blogspot.com.ar

mina el número de hilos simultáneos que el programador de servicio permitirá a la vez. Si el límite es de cuatro, por ejemplo, la **DefaultWorkflowSchedulerService** tomará hasta cuatro hilos del pool del .NET Framework para ejecutar el trabajo. Si cuatro flujos de trabajo ya están en ejecución, otros elementos de trabajo (flujos de trabajo) se sitúan en la cola al final, como hilos de ejecución disponibles.
Es posible determinar el número máximo de flujos de trabajos que pueden estar activos en un momento dado, si se establece un valor como parámetro en el constructor del **DefaultWorkflowSchedulerService** o si se utiliza un archivo de configuración de aplicación (app.config).

A continuación, veremos cómo configurar estos valores mediante la utilización de código y de un archivo de configuración.



www.redserspremium.blogspot.com.ar

FIGURA 011 | Flujo de ejecución de un servicio asincrónico.

Con código (VB):

```
...
Imports System.Workflow.Runtime.Tracking
...
Private Shared maxSimultaneousWorkflows As
Integer = 1
...
workflowRuntime.AddService
(New DefaultWorkflowShcedulerService
(maxSimultaneousWorkflows))
...
```

Con código (C#):

```
...
using System.Workflow.Runtime.Tracking;
...
static int maxSimultaneousWorkflows = 1;
...
workflowRuntime.AddService
(new DefaultWorkflowShcedulerService
(maxSimultaneousWorkflows));
...
```

Con un archivo de configuración:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <section name="HostingWorkflowRuntime"
        type="System.Workflow.Runtime.
        Configuration.WorkflowRuntimeSection,
        System.Workflow.Runtime, Version=
        3.0.00000.0, Culture=neutral,
        PublicKeyToken=31bf3856ad364e35" />
    </configSections>
    <HostingWorkflowRuntime Name="Hosting">
        <CommonParameters/>
        <Services>
            <add type="System.Workflow.
            Runtime.Hosting.DefaultWorkflow
            SchedulerService, System.Work
            flow.Runtime, Version=
            3.0.00000.0, Culture=neutral,
            PublicKeyToken=31bf3856ad364e35"
            maxSimultaneousWorkflows="1"/>
        </Services>
    </HostingWorkflowRuntime>
</configuration>
```

CommitWorkBatch Services

Este tipo de servicio maneja las transacciones usadas por el motor de ejecución para mantener la consistencia entre el estado interno del workflow y las fuentes de almacenamiento externo (puntos de persistencia).

¿Por qué utilizar servicios CommitWork-Batch?

La razón principal de por qué utilizar este tipo de servicios es la de evangelizar el uso de métodos personalizados para el control de errores. Si el servicio DefaultWorkflowCommitWorkBatchService es propietario de la transacción porque se crea una cuando la propiedad Current retorna null (Nothing en Visual Basic), es posible invocar al delegado más de una vez. En consecuencia, se crea una nueva transacción en cada llamada. Un ejemplo frecuente de esto es cuando se manejan problemas intermitentes de redes o de clusters de SQL sobrecargados. Si la llamada al CommitWorkBatchCallback arroja una excepción, WorkflowCommitWorkBatchService no podrá capturar esta excepción, comenzar una nueva transacción e invocar nuevamente al delegado.

www.reduserspremium.blogspot.com.ar

Esto le proporciona un nivel de resistencia tal a la instancia del flujo de trabajo, que de otra forma haría que estos flujos de trabajo finalicen en forma repentina.

Un servicio del tipo WorkflowCommitWorkBatch nos permite manejar los errores que se producen en una transacción.

Cuando un lote de trabajos ha realizado acciones "commit", el motor de tiempo de ejecución realiza llamadas al servicio **CommitWorkBatch**, y pasa un delegado como parámetro para hacer efectiva la labor del proceso de lotes. El motor de tiempo de ejecución aún debe realizar el "commit", pero si se permite que intervenga un servicio el resultado es la personalización del proceso de "commit". A continuación, aprenderemos cómo utilizar los dos servicios **CommitWorkBatch** definidos por Windows Workflow Foundation.

DefaultWorkflowCommitWork BatchService

Este tipo de servicio **CommitWorkBatch** es el servicio que se crea de manera predefinida por Windows Workflow Foundation en el caso de no especificar, de manera explícita, qué tipo utilizar. El propósito principal del servicio **Default-WorkflowCommitWorkBatchService** es crear transacciones cuando no existen, cuando se invoca su **método CommitWorkBatch**.
Si una transacción no existe (esto suele suceder cuando la propiedad **Current** retorna **null**), el **DefaultWorkflowCommitWorkBatchService** debe crearla y establecer su ambiente antes de invocar al delegado **CommitWorkBatchDelegate**. Para hacer esto es necesario envolver la llamada al delegado dentro de un **TransactionScope**.

SharedConnectionWorkflow CommitWorkBatchService

Este tipo de servicio se utiliza para las transacciones de bases de datos que utilizan conexiones compartidas entre diferentes objetos.
Con este tipo de servicio en nuestros flujos de trabajo, les daremos soporte a las tareas por lotes para mantener la integridad de los datos.

Para utilizar el servicio **SharedConnection-WorkflowCommitWorkBatchService** en lugar del servicio **DefaultWorkflowCommit-WorkBatchService** podemos agregarlo mediante la utilización del método **AddService** de nuestra instancia **WorkflowRuntime**. Para esto, es necesario que ésta se instancie y se pase la cadena de conexión a la base de datos por parámetro. Veamos un ejemplo:

C#

```
static void Main(string[] args) {
   string connectionString = " Initial Catalog=
   WorkflowDataStore;Data Source=localhost;
   Integrated Security=SSPI;";

   WorkflowRuntime workflowRuntime = new Work
   flowRuntime();

   workflowRuntime.AddService( new
   SharedConnectionWorkflowCommitWorkBatchService
   (connectionString));
   workflowRuntime.StartRuntime();

   // ...

   workflowRuntime.StopRuntime();
}
```

VB:

```
Shared Sub Main(ByVal args() As String)
   Dim connectionString As String = " Initial
   Catalog=WorkflowDataStore;Data Source=
   localhost;Integrated Security=SSPI;"

   Dim workflowRuntime As New WorkflowRuntime()

   workflowRuntime.AddService(New
```

www.redusers premium.blogspot.com.ar

```
    SharedConnectionWorkflowCommitWorkBatchService
    (connectionString))
    workflowRuntime.StartRuntime()

    ' ...
    workflowRuntime.StopRuntime()
End Sub 'Main
```

También podemos utilizar el archivo de configuración de nuestra aplicación para crear un servicio de este tipo. Para ello, debemos añadir la cadena de conexión en la sección "CommonParameters" dentro del archivo de configuración.

Archivo de configuración:

```
...
  <configSections>
    <section name="WorkflowServiceContainer"
    type="System.Workflow.Runtime.
    Configuration.WorkflowRuntimeSection,
    System.Workflow.Runtime, Version=1.0.0.0,
    Culture=neutral, PublicKeyToken=
    31bf3856ad364e35" />
  </configSections>
  <WorkflowServiceContainer Name="Container
  Name" UnloadOnIdle="true">
    <CommonParameters>
      <add name="ConnectionString" value=
      "Initial Catalog=WorkFlowStore;Data
      Source=localhost;Integrated Security=
      SSPI;" />
    </CommonParameters>
    ...
  </WorkflowServiceContainer>
...
```

Servicios de persistencia

Una gran cantidad de procesos de negocios suelen tardar mucho tiempo antes de completarse (incluso algunas veces meses o años).

Una gran cantidad de procesos de negocios suelen tardar mucho tiempo antes de completarse, e incluso algunas veces suelen demorar meses o años.

Mantener el flujo de trabajo en memoria no sólo es poco práctico debido a las limitaciones de memoria, sino que además impide la escalabilidad de la aplicación porque una instancia se debe procesar en un solo servidor. Muchos de estos flujos de trabajo de larga duración no activan el flujo de trabajo, o la lógica del proceso, y quedan inactivos, esperando la acción del usuario o de otros sistemas.
Si se descarga una instancia inactiva, la aplicación host puede recuperar memoria y permitir la escalabilidad mediante los servidores de procesamiento.

Cuando se producen ciertas condiciones de un flujo de trabajo mientras está en ejecución, el motor de Windows Workflow Foundation utiliza un servicio de persistencia con el objetivo de persistir el estado de la información del flujo de trabajo. Estas condiciones incluyen cosas como las siguientes:

- Cuando se completan las transacciones de actividades **TransactionScopeActivity** y de actividades **CompensatableTransactionScopeActivity**.
- Cuando la instancia de un flujo de trabajo se vuelve inactiva y la propiedad UnloadOnIdle de WorkflowPersistenceService se encuentra en true.

www.reduserspremium.blogspot.com.ar

El motor de tiempo de ejecución del flujo de trabajo determina cuándo debe ocurrir la persistencia. Sin embargo, es necesario un servicio de persistencia para llevar a cabo este tipo de operaciones.

- Cuando una aplicación host en tiempo de ejecución invoca a **System.Workflow.-Runtime.WorkflowInstance.Unload** o **System.Workflow.Runtime. WorkflowInstance.TryUnload** en la instancia de un flujo de trabajo.
- Cuando una instancia de un flujo de trabajo finaliza.
- Cuando se completa una actividad personalizada que utiliza el atributo PersistOnCloseAttribute.

Si una de estas condiciones se cumple y se le añade un servicio de persistencia al motor de tiempo de ejecución, éste recurre a los métodos que ofrece el servicio de persistencia para guardar información sobre el estado de la instancia del flujo de trabajo. Del mismo modo, cuando el motor de flujo de trabajo tiene que restablecer una instancia anterior persistida de un flujo de trabajo, éste invoca los métodos que ofrece el servicio de persistencia para cargar el estado de la información. En otras palabras, el motor de tiempo de ejecución del flujo de trabajo determina cuándo debe ocurrir la persistencia. Sin embargo, un servicio de persistencia es necesario para llevar a cabo este tipo de operaciones.

Es posible dar origen a un nuevo servicio de persistencia si se crea una clase que herede de la clase base abstracta **WorkflowPersistence-Service**. Esta clase contiene cinco métodos abstractos que debemos sobrescribir al momento de crear nuestro servicio de persistencia. En la Tabla 3 podemos ver cuáles son estos métodos y para qué sirven.

Ejemplo en C#:

```
...
  public class MiServicioDePersistencia :
  WorkflowPersistenceService {
    private bool unloadOnIdle = false;

    protected override void SaveWorkflow
    InstanceState(Activity rootActivity, bool
    unlock) {
      // ...
    }

    protected override Activity LoadWork
    flowInstanceState(Guid instanceId) {
```

Tabla 3 | Métodos abstractos de WorkflowPersistenceService

Método	Descripción
LoadCompletedContextActivity	Carga un ámbito específico completado nuevamente en la memoria.
LoadWorkflowInstanceState	Carga una instancia específica de un flujo de trabajo nuevamente en la memoria.
SaveCompletedContextActivity	Graba un ámbito específico completado en un medio de almacenamiento.
SaveWorkflowInstanceState	Graba una instancia específica de un flujo de trabajo en un medio de almacenamiento.
UnlockWorkflowInstanceState	Desbloquea una instancia específica de un flujo de trabajo.

www.reduserspremium.blogspot.com.ar

```
        // ...
      }

      protected override void UnlockWork
      flowInstanceState(Activity state) {
        // ...
      }

      protected override void SaveCompleted
      ContextActivity(Activity activity) {
        // ...
      }

      protected override Activity LoadCompleted
      ContextActivity(Guid activityId, Activity
      outerActivity) {
        // ...

      }

      protected override bool UnloadOnIdle
      (Activity activity) {
        return unloadOnIdle;
      }

    }
...
```

Servicios de seguimiento

Windows Workflow Foundation nos permite realizar el seguimiento del flujo de trabajo, y flujo de trabajo relacionados con la información de una manera consistente, confiable y flexible.

Los servicios de seguimiento de Windows Workflow Foundation fueron diseñados para permitir que los hosts observaran las instancias de flujos de trabajo durante su ejecución y capturaran los eventos que se disparan durante su ejecución. Además, es posible extenderlos y crear así nuestros propios servicios de seguimiento (Custom Tracking Services).

Además, dado que el motor de tiempo de ejecución de Windows Workflow Foundation nos permite agregar varios servicios de tiempo de ejecución durante su vida útil, es posible habilitar múltiples servicios de seguimiento de diferentes tipos simultáneamente. Por ejemplo, Windows Workflow Foundation dispone de un servicio **SqlTrackingService** configurable, que escribe una cantidad de información de seguimiento en una base de datos SQL Server. Otro ejemplo puede ser el de **ConsoleTracking-Service**, que escucha los eventos de un flujo de trabajo y los reproduce en la consola. Ambos servicios se pueden ejecutar en forma simultánea para permitir que los usuarios finales puedan ver esta información cuando ejecuten el flujo de trabajo y cuando depuren el flujo de trabajo durante su desarrollo.

Windows Workflow Foundation contiene varias funciones que permiten el seguimiento de aplicaciones con flujos de trabajo. Gracias a éstas, se producen las siguientes ventajas:

- Se asegura que el seguimiento se produzca de forma coherente.
- Se ofrece escalabilidad y fiabilidad.
- Se permite que los datos del flujo de trabajo se rastreen independientemente de los datos almacenados.
- Se proporciona una ubicación para las consultas de los flujos de trabajo relacionales mediante los medios de almacenamiento.
- Se proporciona la capacidad de consultar sobre el presente y el pasado de los ciclos de vida de los flujos de trabajo, y se determinan posibles rutas de ejecución de la instancia del flujo de trabajo.
- Se proporciona soporte para los cambios programáticos para los servicios de seguimiento.

www.reduserspremium.blogspot.com.ar

Los servicios de seguimiento poseen una funcionalidad que nos permite filtrar la información de seguimiento que más nos interesa y establecer dónde queremos almacenarla (por ejemplo, una base de datos SQL, un archivo físico, etcétera). A esta funcionalidad se la conoce como **Trackin Profiles** o **Perfiles de Seguimiento**.
Podemos crear un perfil de seguimiento mediante el uso de un XML que respete un esquema, o mediante código, si utilizamos el modelo de objetos del servicio de seguimiento. Veamos un ejemplo de cómo crear un perfil de seguimiento con código y con un archivo XML:

C#:

```
static void CreateSimpleTrackingProfile() {
    TrackingProfile miPerfil = new
    Tracking Profile();
    ActivityTrackPoint miPuntoDeSeguimiento =
    new ActivityTrackPoint();
    ActivityTrackingLocation miUbicacion =
    new ActivityTrackingLocation(typeof
    (CodeActivity));
    miUbicacion.MatchDerivedTypes = false;
    miUbicacion.ExecutionStatusEvents.Add
    (ActivityExecutionStatus.Initialized);
    miUbicacion.ExecutionStatusEvents.Add
    (ActivityExecutionStatus.Executing);
    miUbicacion.ExecutionStatusEvents.Add
    (ActivityExecutionStatus.Closed);
    miPuntoDeSeguimiento.MatchingLocations.
    Add(miUbicacion);
    miPerfil.ActivityTrackPoints.
    Add(miPunto DeSeguimiento);
    miPerfil.Version = new Version
    ("1.0.0.0");
    TrackingProfileSerializer
    tracking ProfileSerializer =
    new TrackingProfile Serializer();
    StringBuilder trackingProfileString = new
    StringBuilder();
    using (StringWriter writer = new
    String Writer(trackingProfileString,
    CultureInfo.InvariantCulture))
    {
        trackingProfileSerializer.Serialize
        (writer, miPerfil);
        Console.WriteLine(writer.ToString());
    }
}
```

XML:

```
<?xml version="1.0" encoding="utf-16"
standalone="yes"?>
<TrackingProfile xmlns="http://schemas.
microsoft.com/winfx/2006/workflow/trackin
gprofile" version="1.0.0.0">
  <TrackPoints>
    <ActivityTrackPoint>
      <MatchingLocations>
        <ActivityTrackingLocation>
          <Activity>
            <Type>System.Workflow.
            Activities.CodeActivity,
            System.Workflow.Activities,
            Version=3.0.0.0, Culture=
            neutral, PublicKeyToken=
            31bf3856ad364e35</Type>
            <MatchDerivedTypes>false
            </MatchDerivedTypes>
          </Activity>
          <ExecutionStatusEvents>
            <ExecutionStatus>Initialized
            </ExecutionStatus>
            <ExecutionStatus>Executing
            </ExecutionStatus>
            <ExecutionStatus>Closed
            </ExecutionStatus>
          </ExecutionStatusEvents>
        </ActivityTrackingLocation>
      </MatchingLocations>
    </ActivityTrackPoint>
  </TrackPoints>
</TrackingProfile>
```

www.reduserspremium.blogspot.com.ar

Tipos de Workflows

En las próximas líneas nos referiremos a los tipos de flujos de trabajo, de manera más técnica.

Anteriormente en este capítulo hicimos referencia a los tipos de flujos de trabajo o workflows, que son flujos de trabajo de personas y de sistemas, y los comparamos dentro de un ámbito cotidiano.

Además de referirnos a los tipos de flujos de trabajo, veremos cómo podemos crearlos mediante la utilización de la herramienta **Designer** de Windows Workflow Foundation.

Workflows Secuenciales o Sequential Workflows

Los flujos de trabajo secuenciales, como su mismo nombre lo indica, son aquellos que comienzan en un punto y siguen una secuencia preestablecida de forma secuencial (de uno en uno), hasta que finalizan su última actividad. Sin embargo, éstos no siempre son completamente secuenciales, debido a que pueden tener llamadas de eventos externos, incluir actividades en paralelo, etcétera, y así hacer que el orden exacto de ejecución varíe ligeramente.

A continuación, vamos a aprender cómo crear y probar un workflow del tipo secuencial. Para ello creamos un nuevo proyecto en **VisualStudio** del tipo **Sequential Workflow Console Application**, y como nombre le pondremos **MiWorkflowSecuencial**.

Ahora podemos empezar a diseñar nuestro flujo de trabajo. Simplemente abrimos el flujo de trabajo con el nombre **Workflow1.cs** y podemos empezar a arrastrar las actividades desde la barra de herramientas hacia nuestro flujo de trabajo.
Para nuestro caso, sólo arrastraremos la actividad del tipo **IfElseActivity** y luego sobre cada **IfElseBranchActivity** arrastraremos un **CodeActivity**.

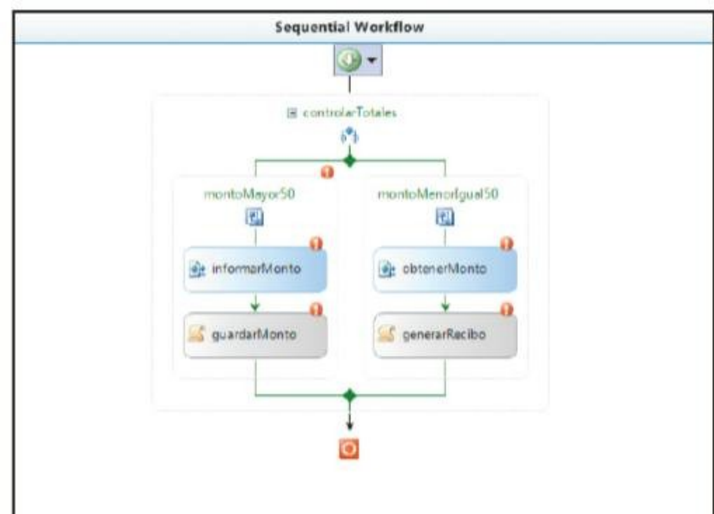
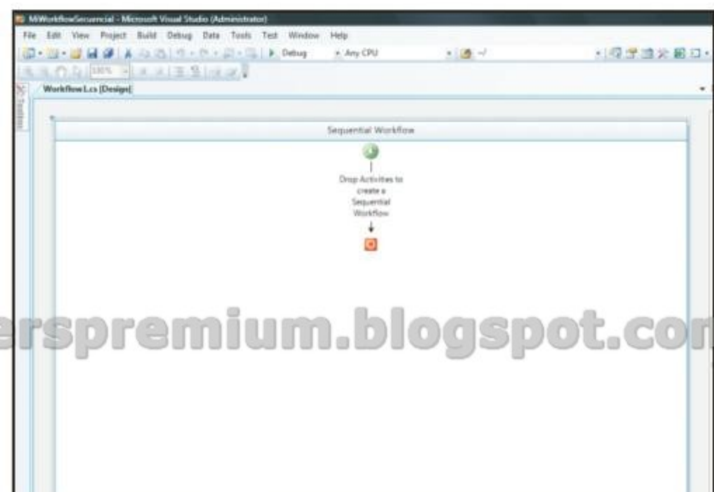


FIGURA 012 | Ejemplo de un flujo de trabajo del tipo secuencial.



www.reduserspremium.blogspot.com.ar

FIGURA 013 | Podemos crear una aplicación de consola que hospede un workflow en un simple paso.

La arquitectura de un flujo de trabajo del tipo secuencial es similar a la de una aplicación desarrollada basada en procedimientos.

Ahora sólo nos resta crear los manejadores del flujo de trabajo. Para ello seleccionamos la actividad IfElseBranchActivity1 y en la ventana de propiedades establecemos la propiedad **Condition** con el valor **Code Condition**. Este valor lo seleccionamos de la lista desplegable para esa propiedad. De esta forma establecemos que la condición será manejada mediante código. Después de esto, prodecemos a expandir la propiedad **Condition** (hacemos clic sobre el signo + que se encuentra a su izquierda) y escribimos el nombre del método que contendrá nuestra lógica. Para nuestro ejemplo usaremos el nombre "**Validar Condición**" y al presionar ENTER, VisualStudio nos mostrará la vista del código de este workflow, donde estableceremos la propiedad **Result** del parámetro "e" en true. Éste indica que el resultado de la condición se cumple.

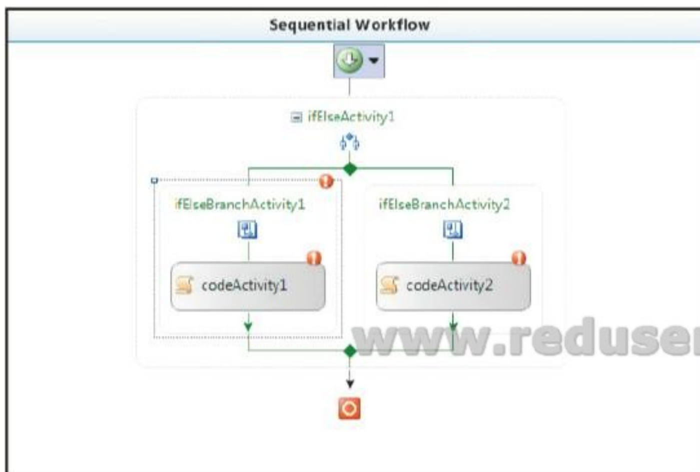


FIGURA 014 | Ejemplo de cómo debería estar compuesto un flujo de trabajo secuencial simple.

C#:

```
namespace MiWorkflowSecuencial {
  public sealed partial class Workflow1 :
  SequentialWorkflowActivity {
    public Workflow1() {
      InitializeComponent();
    }

    private void ValidarCondicion(object
    sender, ConditionalEventArgs e) {
      e.Result = true;
    }
  }
}
```

VB:

```
Public class Workflow1
    Inherits SequentialWorkflowActivity

  Private Sub ValidarCondicion(ByVal sender
  As System.Object, ByVal e As System.
  Workflow.Activities.ConditionalEventArgs)
    e.Result = True
  End Sub
End Class
```

De la misma forma, estableceremos los manejadores para las dos actividades del tipo **CodeActivity**. Para ello estableceremos la propiedad ExecuteCode de estas actividades. Entonces, debemos escribir SiEsSi para **CodeActiviy1** y SiEsNo para **CodeActivity2**. Estos métodos que se generarán nos servirán para ejecutar código cuando la instancia del flujo de trabajo pase por estas actividades.

Ahora solamente escribiremos unas líneas de código para escribir algunos mensajes en la ventana de consola con el objetivo de saber por cuál actividad pasó la instancia.
Para ello, debemos escribir la siguiente línea de código en los dos últimos métodos generados:

www.reduserspremium.blogspot.com.ar

C#:

```
Console.WriteLine("La instancia está en la
Actividad SiEsSi");
```

VB:

```
Console.WriteLine("La instancia está en la
Actividad SiEsSi")
```

Y por último, dentro de nuestra clase **Program** (en el caso de C#) o del módulo **Module1** (en el caso de VB), agregamos la siguiente línea de código justo antes de la finalización del método **Main**. De esta forma, ya estamos listos para probar nuestro flujo de trabajo secuencial.

C#:

```
...
    static void Main(string[] args) {
      //
      // líneas generadas automáticamente
      por VS.
      //
      Console.WriteLine("Proceso finalizado,
      presione cualquier tecla...");
      Console.ReadKey();
    }
...
```

VB:

```
...
    Shared Sub Main()
      ''
      '' líneas generadas automáticamente
      por VS.
      ''
      Console.WriteLine("Proceso finalizado,
      presione cualquier tecla...")
      Console.ReadKey()
    End Sub
...
```

Aprendimos cómo crear un flujo de trabajo secuencial. Este tipo de flujo de trabajo nos servirá para implementarlo en los flujos de trabajo de máquina de estados, que son más flexibles y capaces de abarcar un terreno más amplio que los flujos de trabajo secuenciales, incluso en lo que a interacción humana se refiere.

Workflows de Máquina de Estados o State Machine Workflows

Las máquinas de estados están con nosotros desde hace mucho tiempo. Son muy populares en los sistemas reactivos, como los video juegos o la robótica. Los diseñadores utilizan las máquinas de estados para moldear sistemas que utilicen estados, eventos y transiciones.

Un **estado** representa una situación o una circunstancia, un **evento** representa un estímulo externo y una **transición** se refiere al cambio de un estado específico.
Para entender estos conceptos, citaremos este ejemplo:

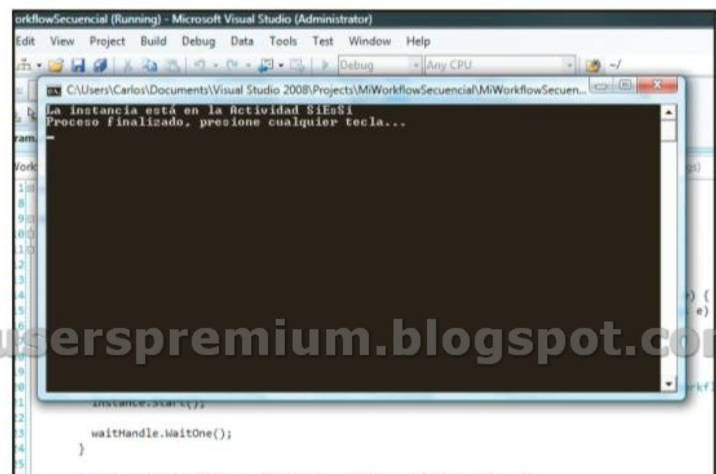


FIGURA 015 | Ejemplo de ejecución de una aplicación de consola con un flujo de trabajo secuencial.

www.reduserspremium.blogspot.com.ar

Una aplicación que se encarga de habilitar y deshabilitar un puerto en una computadora incorporaría estos tres conceptos y tendría dos **estados**: habilitado y deshabilitado; un **evento**, que sería un botón que cambiaría de estados y finalmente la **transición** sería cuando el estado habilitado cambiara a deshabilitado.

Estos conceptos son utilizados para el desarrollo de flujos de trabajo de este tipo en Windows Workflow Foundation.

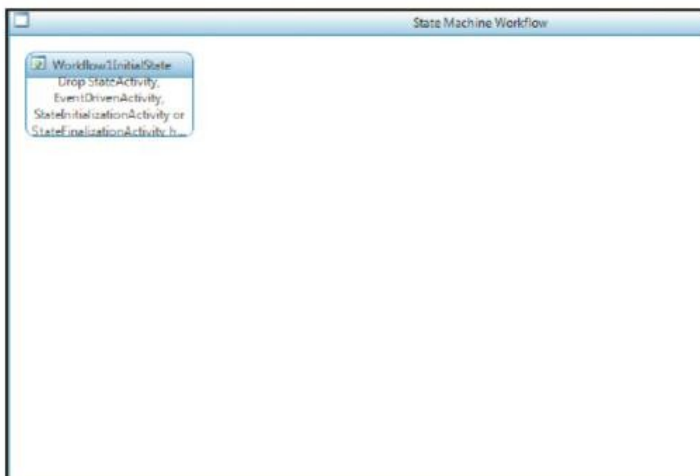


FIGURA 016 | Al crear un nuevo proyecto de flujo de trabajo de máquina de estados, solamente contaremos con el estado inicial en el flujo de trabajo.

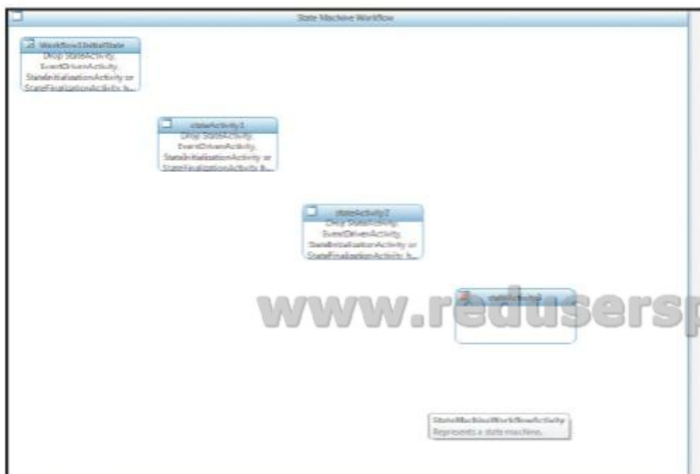


FIGURA 017 | Al establecer un estado de finalización, éste cambia su icono de referencia.

En WWF, la actividad **StateActivity** representa un estado en el flujo de trabajo. Cuando se dispara un evento, el flujo de trabajo realiza transiciones entre las actividades de estados. El flujo de trabajo puede tener un estado de finalización, y el flujo de trabajo finaliza cuando realiza una transición a un estado de finalización.

Un **EventDrivenActivity** representa un manejador de eventos en una máquina de estados. Nosotros podemos colocar estas actividades dentro de las actividades de Estados para representar los eventos de un estado. Dentro de este tipo de actividad podemos colocar actividades secuenciales, que se ejecutarán cuando un evento se dispare. La última actividad dentro de esta secuencia debe ser una actividad **SetStateActivity**, que especifica la transición al próximo estado.

Veamos un ejemplo de un flujo de trabajo de máquina de estados. Para ello recurriremos nuevamente a VisualStudio, donde crearemos un nuevo proyecto del tipo **State Machine Workflow Console Application**.
Este flujo de trabajo constará de cuatro estados, un inicial, dos intermedios y uno de finalización.

Al crear el proyecto en VisualStudio, solamente veremos en nuestro flujo de trabajo un estado, que será el inicial. Por lo tanto, debemos agregar los tres estados que nos restan. Para ello arrastramos las actividades **State** dentro de nuestro flujo de trabajo desde la barra de herramientas.

www.reduserspremium.blogspot.com.ar

Ahora, definiremos nuestro estado de finalización. Para ello debemos seleccionar nuestro flujo de trabajo y en la ventana de propiedades establecemos la propiedad **CompletedStateName** con el último estado que

agregamos a nuestro workflow, en nuestro caso **stateActivity3**. También podemos usar el menú contextual de la actividad **stateActivity3** para realizar esta operación. Sólo basta con seleccionar "**Set as Completed State**".

Ahora, para cada uno de los tres primeros estados (Workflow1InitialState, stateActivity1 y stateActivity2) debemos crear las actividades del tipo **EventDriven**. Para ello, simplemente seleccionamos la opción "Add EventDriven" del menú contextual de la actividad y dentro de esta actividad debemos agregar las siguientes actividades:

La primera debe implementar la interfaz **IeventActivity**, o sea, debe ser una actividad de eventos. Por lo tanto, agregamos una **DelayActivity** como primera actividad de la actividad **EventDriven**. A continuación agregamos una actividad del tipo **CodeActivity**, y finalmente una actividad del tipo **SetState**, que se encargará de las transiciones. Para esto debemos arrastrar estas actividades desde la barra de herramientas hacia el flujo de trabajo. Para la actividad del tipo **CodeActivity** debemos establecer su manejador de la misma manera que lo hicimos en el ejemplo anterior. Tenemos que establecer el nombre del método en la propiedad **ExecuteCode**. Para nuestro ejemplo especificaremos el nombre **Manejador1**, **Manejador2**, **Manejador3** para cada una de las actividades **CodeActivity**. En la Figura 18 podemos ver el aspecto que deben tener nuestras actividades del tipo **EventDriven**.

A continuación, vamos a definir la transición entre los estados. Para ello debemos editar las propiedades de cada actividad del tipo **SetState** que incluimos en las actividades **EventDriven**. Éste es un proceso de asociación, por lo que es muy sencillo de realizar.

Un estado representa una situación o una circunstancia, un evento representa un estímulo externo y una transición se refiere al cambio de un estado específico. Sobre estos conceptos se basa el desarrollo de flujos de trabajos de máquinas de estados.

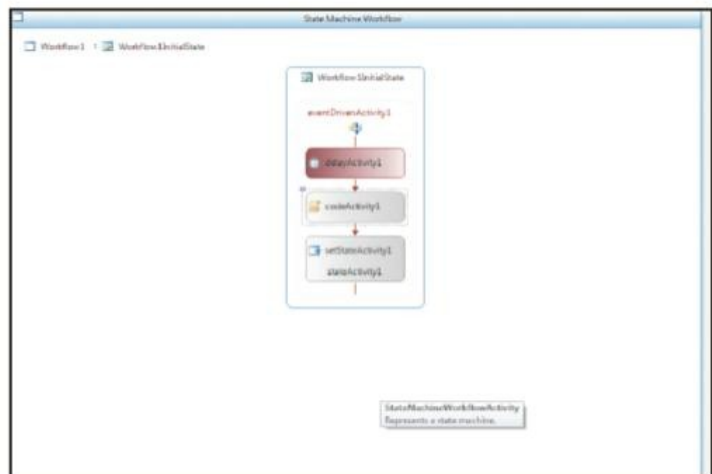


FIGURA 018 | Aspecto del contenido de una actividad EventDriven en un flujo de trabajo.

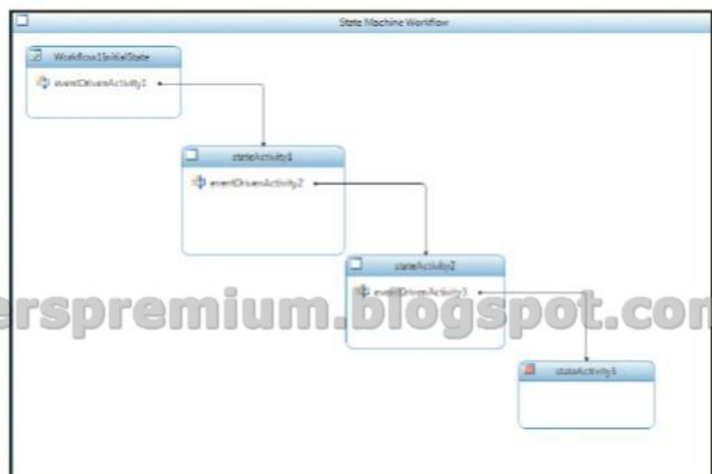


FIGURA 019 | Una vez establecidas las transiciones, el diseñador de WF nos muestra de forma gráfica esta relación.

www.reduserspremium.blogspot.com.ar

La propiedad TargetStateName debe establecerse con el nombre del siguiente estado arrastrado al flujo de trabajo, o sea, StateActivity1. Para SetStateActivity2, el valor debe ser StateActivity2 y lo mismo con SetStateActivity3.

Simplemente debemos seleccionar a qué estado debe cambiar para continuar con el flujo de trabajo. La propiedad **TargetStateName** sirve para este propósito. Al desplegar la lista, VisualStudio nos mostrará todos los estados que existen en nuestro flujo de trabajo, seleccionamos el que corresponda y eso es todo.
Para nuestro caso, la propiedad **TargetStateName** de la actividad **SetStateActivity1** (que se encuentra dentro de EventDrivenActivity1, que a su vez está en el estado Workflow1-InitialState) debe establecerse con el nombre del segundo estado arrastrado al flujo de trabajo, o sea, **StateActivity1**. Para la actividad **SetStateActivity2**, el valor de la propiedad debe ser **StateActivity2**, y para la actividad **SetStateActivity3**, el valor de la propiedad debe ser **StateActivity3**.

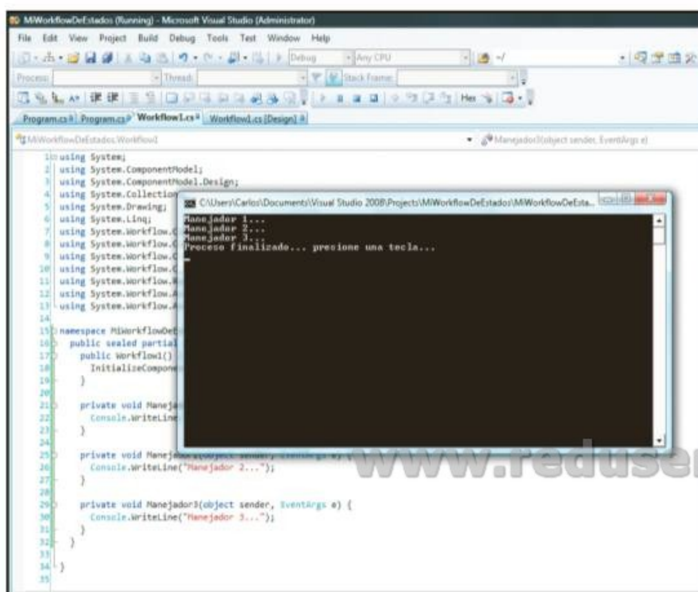


FIGURA 020 | Ejemplo de ejecución de una aplicación de consola con un flujo de trabajo de máquina de estados.

En este punto ya estamos en condiciones de probar nuestro flujo de trabajo. Sin embargo, sólo aparecería una ventana de comandos que se abre y se cierra rápidamente. Esto es claro porque no contiene lógica aplicada a nuestro flujo de trabajo. Lo que haremos a continuación será escribir algunas líneas de código para ver por cuál actividad pasa la instancia de nuestro flujo de trabajo.
Sobre los manejadores de las actividades **CodeActivity**, escribiremos lo siguiente:

C#:

```
Console.WriteLine("Manejador 1...");
```

VB:

```
Console.WriteLine("Manejador 1...")
```

Y por último, como en el ejemplo anterior, escribiremos las siguientes líneas en el método **Main** de nuestra aplicación:

C#:

```
Console.WriteLine("Proceso finalizado...
presione una tecla...");
Console.ReadKey();
```

VB:

```
Console.WriteLine("Proceso finalizado...
presione una tecla...")
Console.ReadKey()
```

www.reduserspremium.blogspot.com.ar

Ahora sí estamos en condiciones de ejecutar nuestra aplicación y probar nuestro flujo de trabajo de máquina de estados.

USERS

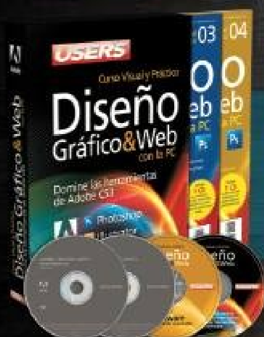
CURSOS INTENSIVOS

CURSOS.REDUSERS.COM

Los temas más importantes del universo de la tecnología desarrollados con la mayor profundidad y con un despliegue visual de alto impacto: Explicaciones teóricas, procedimientos paso a paso, videotutoriales, infografías y muchos recursos mas.

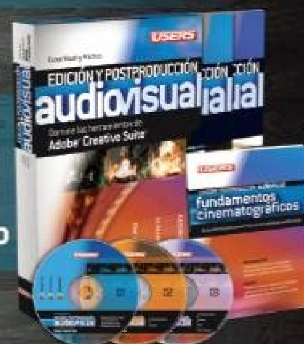
Brinda las habilidades necesarias para planificar, instalar y administrar redes de computadoras de forma profesional. Basada principalmente en tecnologías Cisco, es una obra actual, que busca cubrir la necesidad creciente de formar profesionales.

- 25 Fascículos
- 600 Páginas
- 3 CDs / 1 Libro



- 25 Fascículos
- 600 Páginas
- 4 CDs

Curso para dominar las principales herramientas del paquete Adobe CS3 y conocer los mejores secretos para diseñar de manera profesional. Ideal para quienes se desempeñan en diseño, publicidad, productos gráficos o sitios web.



Obra teórica y práctica que brinda las habilidades necesarias para convertirse en un profesional en composición, animación y VFX (efectos especiales).

- 25 Fascículos
- 600 Páginas
- 2 CDs / 1 DVD / 1 Libro

- 26 Fascículos
- 600 Páginas
- 2 DVDs / 2 Libros

Obra ideal para ingresar en el apasionante universo del diseño web y utilizar Internet para una profesión rentable. Elaborada por los máximos referentes en el área, con infografías y explicaciones muy didácticas.

www.reduserspremium.blogspot.com.ar

Llegamos a todo el mundo con OCA * y DHL **

usershop@redusers.com +54 (011) 4110-8700

usershop.redusers.com.ar

** Válido en todo el mundo excepto Argentina. * Sólo válido para la República Argentina

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

23

Windows Workflow Foundation

Los componentes y su arquitectura / Introducción a diseñadores de Workflows

Servicios de Runtime

www.reduserspremium.blogspot.com.ar

Scheduling / CommitWorkBatch / Persistence & Tracking services

ISBN 978-987-1347-43-8