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

24

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Creación de Workflows

Tipos de flujos de trabajo / Reglas y condiciones / Políticas / Tipos de actividades predefinidas

www.redusers-premium.blogspot.com.ar

ISBN 978-987-1347-43-8
00024
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

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

Policies

Como ya hemos visto en los capítulos anteriores, una regla es un conjunto de expresiones condicionales que nos permiten evaluar ciertos valores para determinar si un resultado es verdadero o falso. En este capítulo profundizaremos un poco más acerca de las reglas y la relación que guardan con los Conjuntos de Reglas o **RuleSet**, usado por la actividad **Policy**.

La actividad Policy y los Conjuntos de Reglas

La actividad llamada **Policy** nos permite evaluar un **RuleSet** en un flujo de trabajo. Un **RuleSet** es un conjunto de reglas que pueden estar relacionadas entre sí o no.

Ahora veamos en la práctica cómo podemos utilizar esta actividad dentro de nuestros flujos de trabajo en Workflow Foundation. Para comenzar, abramos Visual Studio.NET. Luego crearemos un nuevo proyecto llamado **PolicyWF** de tipo **Sequential Workflow Console Application**, tal como lo muestra la Figura 38.

Ya que deseamos mostrar la funcionalidad de la actividad **Policy** junto con la evaluación de su **RuleSet** relacionado, necesitamos escribir un poco de funcionalidad para nuestro flujo de trabajo. Sin embargo, ésta es una tarea muy sencilla: vamos a crear cuatro variables públicas del tipo int llamadas a, b, c y d. Estas variables (comúnmente denominadas "campos") le permitirán al **RuleSet** evaluar sus reglas y su posible revaluación. Muy bien, manos a la obra. Hagamos clic en el botón "View Code" de la ventana Solution Explorer de Visual Studio.NET para desplegar la vista de código de nuestro flujo de trabajo. Posteriormente escribamos el código para crear esas variables a nivel de clase:

> Dos reglas o más están relacionadas entre sí cuando leen o actualizan el mismo campo o la misma propiedad de alguna clase.

C#:

```
int a;
int b;
int c;
int d;
```

VB:

```
Dim a As Integer
Dim b As Integer
Dim c As Integer
Dim d As Integer
```

Ahora necesitamos asignarle un valor a cada variable para que las reglas en el **RuleSet** que vamos a crear lean esos valores. Esto lo podemos hacer inmediatamente después de la invocación al método **InitializeComponent** en el constructor de la clase.

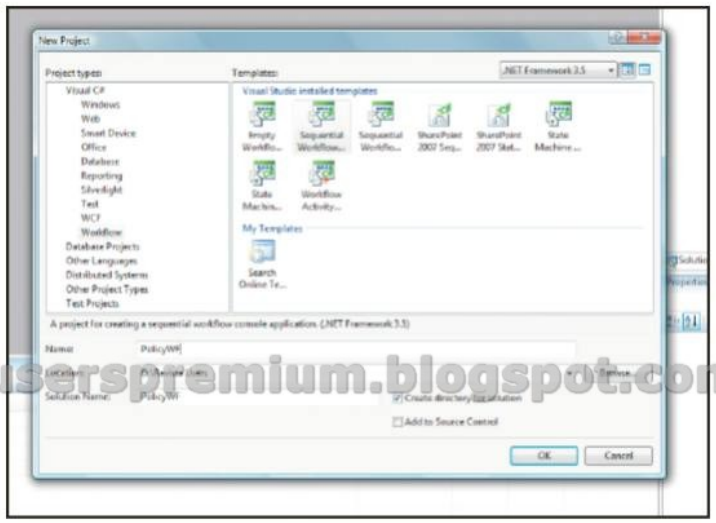


FIGURA 038 | VisualStudio 2005 y 2008 nos permiten crear proyectos de WWF prácticamente funcionales.

www.reduserspremium.blogspot.com.ar

C#:

```
a = 10;
b = 1;
c = 5;
d = 1;
```

VB:

```
a = 10
b = 1
c = 5
d = 1
```

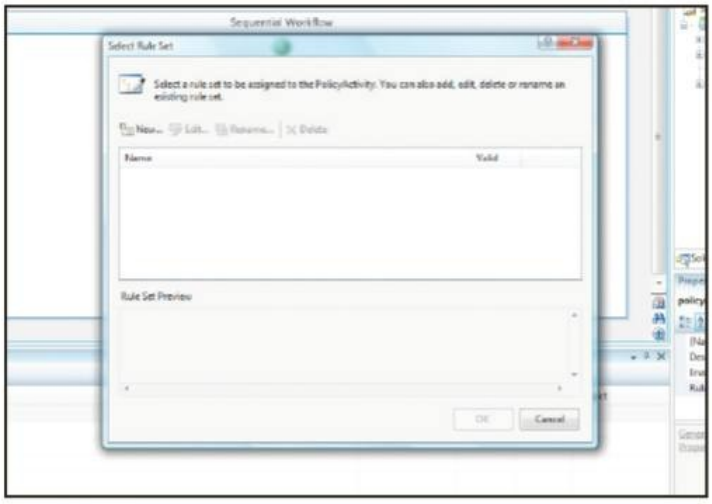


FIGURA 039 | Asociar un conjunto de reglas es muy sencillo con los asistentes de VisualStudio.

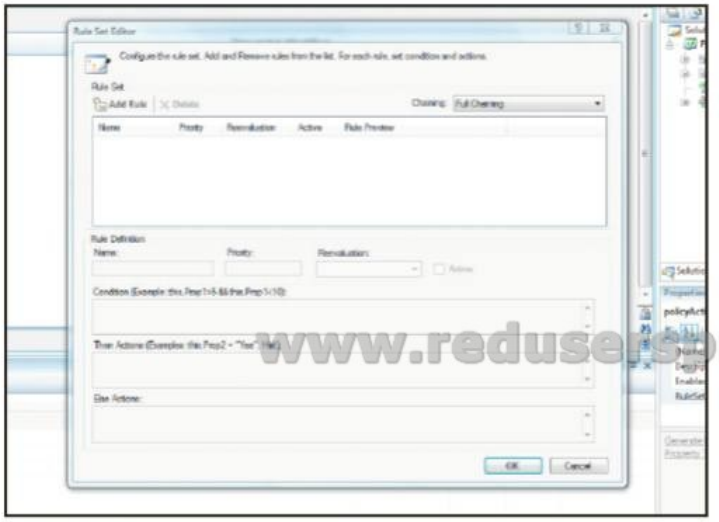


FIGURA 040 | La edición de los conjuntos de reglas es una tarea sencilla si utilizamos VisualStudio.

El siguiente paso es seleccionar la actividad **Policy** del **Toolbox** y arrastrarla al diseñador de Workflow Foundation. Cuando la actividad se coloca en el diseñador, automáticamente Visual Studio.NET le asigna un nombre, que en nuestro caso es **policyActivity1**. Asimismo, esta actividad muestra un signo de admiración de color rojo. Esto significa que la actividad aún no se ha configurado (tal y como sucede al configurar una actividad del tipo **IfElseActivity**, como vimos en los capítulos anteriores).

Como podemos apreciar en la ventana de propiedades, la actividad **policyActivity1** tiene una propiedad llamada **RuleSetReference**, que indica el nombre del **RuleSet** que deseamos asociar a esa actividad.
En este caso acabamos de crear un proyecto nuevo y no tenemos un **RuleSet** definido. Esto lo podemos hacer en el cuadro de diálogo en la llamada **Select Rule Set**. Ésta se despliega al hacer clic en el botón que tiene tres puntos "…" en la propiedad **RuleSetReference** de **policyActivity1**, tal como lo muestra la Figura 39.

Este cuadro de diálogo nos permite crear, modificar o eliminar los Conjuntos de Reglas que hayamos definido dentro de nuestro proyecto. Debido a que no tenemos nada definido aún, una buena idea es hacer clic en el botón "**New…**" para poder crear un nuevo **RuleSet**. Al hacer clic en este botón, Visual Studio.NET despliega el cuadro de diálogo con la llamada "**Rule Set Editor**", que nos permite establecer las reglas que componen ese **RuleSet**, su configuración de encadenamiento o **Chaining** (que explicaremos más adelante), su prioridad, su comportamiento de revaluación, la expresión condicional, las acciones a ejecutar si esa expresión es verdadera (THEN) y las acciones a ejecutar si esa expresión resulta ser falsa (ELSE). Podemos apreciar este cuadro de diálogo en la Figura 40.

www.redusers-premium.blogspot.com.ar

Creación de actividades personalizadas

En este capítulo aprenderemos los pasos básicos para crear una actividad personalizada dentro de WWF.

Actividades personalizadas

Windows WorkFlow Foundation provee varias actividades preestablecidas, como **CodeActivity**, **CompensateActivity**, **DelayActivity**, etcétera. Sin embargo, esas actividades difícilmente pueden contemplar todas las diferentes actividades de negocio o técnicas que necesitemos implementar.
Es por eso que WWF nos permite crear nuestras propias actividades al heredar de diferentes clases base (ver Tabla 8, donde sólo se listan las más importantes).

En este capítulo crearemos una actividad que nos permita realizar el envío de un correo electrónico. Para esto, sólo necesitamos heredar de la clase base **Activity** ya que ésta nos provee toda la funcionalidad básica necesaria.

Para crear la actividad seguiremos los siguientes pasos:

- Abrir Visual Studio 2008.
- En el menú seleccionar **File/New /Project**.
- En **Project Types** seleccionar Workflow.
- En la lista de plantillas seleccionar **Workflow Activity Library**.
- En Name escribir **EnviarCorreoActivity**.
- En el Solution Explorer seleccionar **Activity1**. Hacer clic con el botón derecho del mouse y seleccionar **Rename**. En el nuevo nombre escribir **EnviarCorreoActivity.cs** y presionar **ENTER**.
- En la advertencia haga clic sobre **Yes**.
- En la clase renombrada haga clic nuevamente con el botón derecho del mouse y seleccione **View Code**.
- Cambiar la clase de la que se hereda **SequenceActivity** por **Activity**.

En este punto debemos de tener una clase similar a la siguiente:

C#:

```
using System;
using System.ComponentModel;
```

Tabla 8 | Clases base para crear una actividad personalizada

Clase base	Descripción
Activity	Todas las actividades heredan de esta clase.
SequenceActivity	Se utiliza para crear una actividad compuesta que contiene actividades hijas que se ejecutan en forma secuencial.
StateActivity	Se utiliza dentro de un StateMachine WorkFlow y expone eventos que permiten la inicialización de la actividad a un estado seleccionado.
CompositeActivity	Permite crear una actividad compuesta sin la posibilidad de tener actividades hijas.

www.radusoftpremium.blogspot.com.ar

```
using System.ComponentModel.Design;
using System.Collections;
using System.Drawing;
using System.Linq;
using System.Workflow.ComponentModel.
Compiler;
using System.Workflow.ComponentModel.
Serialization;
using System.Workflow.ComponentModel;
using System.Workflow.ComponentModel.Design;
using System.Workflow.Runtime;
using System.Workflow.Activities;
using System.Workflow.Activities.Rules;

namespace EnviarCorreoActivity {
  public partial class EnviarCorreoActivity :
  Activity {
    public EnviarCorreoActivity() {
      InitializeComponent();
    }
  }
}
```

Para realizar el envío de correo electrónico necesitamos que la actividad exponga 4 propiedades dependientes "Para" "De" "Asunto" y "Cuerpo". Para ello utilizaremos el método **Register** de la clase **DependencyProperty**:

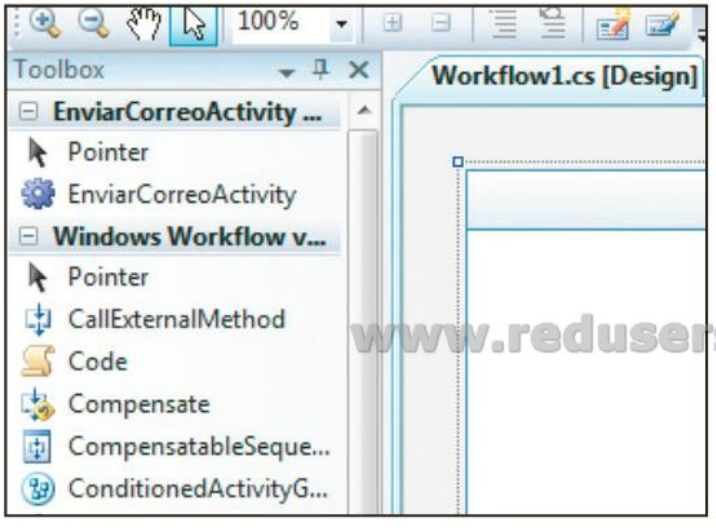


FIGURA 041 | Actividad dentro de la barra de herramientas.

C#:

```
...
    public static DependencyProperty
    DeProperty = DependencyProperty.
    Register("De", typeof(string), typeof
    (EnviarCorreoActivity));
    public static DependencyProperty
    ParaProperty = DependencyProperty.
    Register("Para", typeof(string),
    typeof(EnviarCorreoActivity));
    public static DependencyProperty
    AsuntoProperty = DependencyProperty.
    Register("Asunto", typeof(string),
    typeof(EnviarCorreoActivity));
    public static DependencyProperty
    CuerpoProperty = DependencyProperty.
    Register("Cuerpo", typeof(string),
    typeof(EnviarCorreoActivity));
...
```

Ahora crearemos las propiedades físicas que se encargarán de realizar la persistencia o la obtención de la información proporcionada. Para ello hay que utilizar los métodos **SetValue** y **GetValue**, y la clase base.

C#:

```
...
    public string De {
      get { return Convert.ToString(base.
      GetValue(DeProperty)); }
      set { base.SetValue(DeProperty, value);}
    }

    public string Para {
      get { return Convert.ToString(base.
      GetValue(ParaProperty)); }
      set { base.SetValue(ParaProperty,
      value); }
    }

    public string Asunto {
      get { return Convert.ToString(base.
```

www.reduserspremium.blogspot.com.ar

```
        GetValue(AsuntoProperty)); }
        set { base.SetValue(AsuntoProperty,
        value); }
    }

    public string Cuerpo {
        get { return Convert.ToString(base.
        GetValue(CuerpoProperty)); }
        set { base.SetValue(CuerpoProperty,
        value); }
    }
...
```

Por último, sobrescribiremos el método **Execute** de la clase base. Este método se llama cuando la actividad es requerida para su ejecución. Dentro de este método realizaremos el envío del correo electrónico.
Como resultado de la ejecución regresaremos el valor **Closed** sólo en caso de que el envío sea exitoso. En caso contrario será **Faulting**. Este valor se toma de la enumeración **ActivityExecutionStatus**:

C#:

```
...
    protected override ActivityExecution
    Status Execute(ActivityExecutionContext
    executionContext) {
        try {
            System.Net.Mail.SmtpClient smtpClient
            = new System.Net.Mail.SmtpClient
            ("smtp.servidorsmtp.com");
            smtpClient.Credentials = new
            System.Net.NetworkCredential
            ("nombreDeUsuario", "Constraseña");
            smtpClient.Send(De, Para, Asunto,
            Cuerpo);
            return ActivityExecutionStatus.Closed;
        } catch (Exception ex) {
            string logInterno = ex.Message;
            return ActivityExecutionStatus.
            Faulting;
        }
    }
...
```

La clase base **Activity** proporciona la funcionalidad base para crear cualquier actividad personalizada. Para probar nuestra actividad crearemos un flujo de trabajo del tipo **Sequential Workflow Console Application** y agregaremos una referencia a este proyecto de nuestra actividad personalizada. De esta forma, podremos ver nuestra actividad en la barra de herramientas de VisualStudio (ver Figura 41), y arrastrarla a nuestro flujo de trabajo secuencial (ver Figura 42).

De esta forma, ya hemos creado nuestra primera actividad personalizada. Sólo basta con especificar las propiedades e iniciar el flujo para poder probar que la actividad se ejecuta de manera correcta.

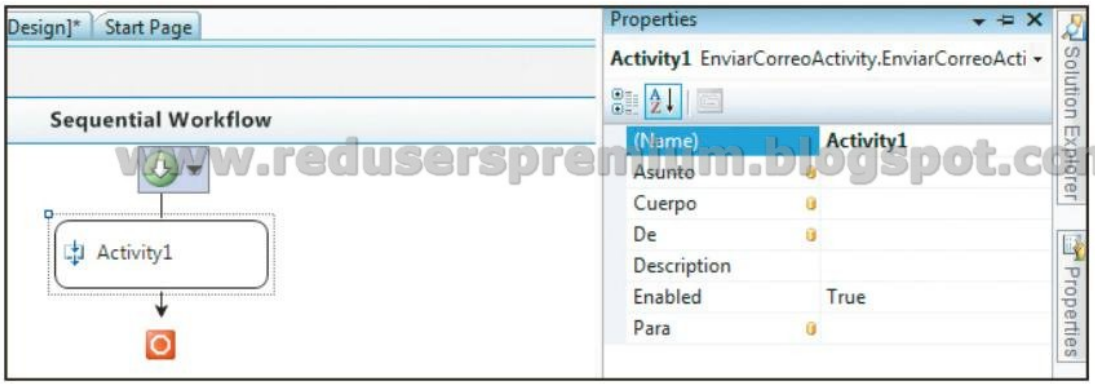


FIGURA 042 | La ventana de propiedades muestra las propiedades definidas para nuestra actividad.

www.reduserspremium.blogspot.com.ar

Compensación en Workflows

Veremos qué son y cómo se utilizan las transacciones, como medio para asegurarnos de que la información se haya actualizado.

Transacciones

Con frecuencia, en nuestras aplicaciones necesitamos incorporar algún tipo de mecanismo que nos asegure que los datos que usamos se hayan grabado de manera correcta y consistente. Bienvenidos a las Transacciones, que en el contexto de WF las podemos utilizar como medio para asegurarnos de que efectivamente la información se haya actualizado tal como esperamos y en caso de alguna falla, la información no quede incompleta o inconsistente.

En WF podemos incorporar dos tipos de transacciones en nuestros flujos de trabajo: transacciones del tipo 2PC (**Two Phase Commit**) y transacciones compensables. En este capítulo explicaremos y mostraremos el uso de cada una de ellas para nuestras aplicaciones.

La Atomicidad nos asegura que todas las operaciones en una transacción se realicen todas o no se realice ninguna.

Una transacción se puede definir como una unidad de trabajo donde se ejecutan exitosamente todas las tareas que se incluyen en la transacción o no se ejecuta ninguna.

Transacciones Two Phase Commit

Este tipo de transacción es aquella que está coordinada mediante un administrador, cuyo objetivo es determinar si las acciones y los datos relacionados con la transacción se deben aplicar en un momento específico. Ese momento lo determina el administrador al tomar en cuenta si todos los recursos relacionados en la transacción "votan" para que ésta continúe o se deshaga en su totalidad. Las transacciones de esta categoría deben tener las propiedades **ACID**, por sus siglas en inglés: **Atomicity**, **Consistency**, **Isolation** y **Durability**.

La prueba del "Ácido" para las transacciones

Como describimos en el párrafo anterior, las transacciones del tipo Two Phase Commit deben tener ciertas propiedades para que su comportamiento sea el esperado y funcionen correctamente. Estas propiedades las explicaremos a continuación:

Atomicidad (Atomicity): esta propiedad indica que todas las operaciones relacionadas con la transacción se han ejecutado en su totalidad o no se han ejecutado, es decir, ésta es la propiedad que nos asegura que un proceso de actualización de datos no puede quedar a medias y hacer que nuestros datos queden en un estado incorrecto.
Un ejemplo donde podemos apreciar esta propiedad es cuando realizamos un retiro de

www.reduserspremium.blogspot.com.ar

dinero de un cajero automático: solicitamos cierta cantidad y esa cantidad se nos da en efectivo.

Ahora bien, imaginemos el caso en el que la transacción no fuera atómica y que ocurriese alguna falla, por ejemplo, que el cajero automático no tuviera dinero suficiente para nuestro retiro. Terminaríamos con un saldo menor e incorrecto en nuestra cuenta de ahorros pero con nada de dinero en efectivo en las manos. O tal vez el pago de un servicio en línea por medio de una tarjeta de crédito. En fin, ejemplos podríamos mencionar muchos. Gracias a la propiedad de atomicidad en las transacciones nos aseguramos de que en el caso de error de los ejemplos anteriores los datos relacionados no se vean afectados.

Consistencia (Consistency): esta propiedad indica que los datos envueltos en la transacción se deben mantener consistentes. Si tomamos como ejemplo una base de datos relacional, esto indica que la transacción no debe romper ninguna restricción que haya sido declarada. Tal es el caso de una transacción que requiere insertar un registro en dos tablas con una relación padre-hijo, por ejemplo las tablas Cliente y Factura. Si la transacción inserta exitosamente un registro en la tabla Factura pero falla al insertar el registro correspondiente en la tabla Cliente, la integridad de la base de datos se vería afectada y la transacción –gracias a la propiedad de Consistencia- se cancela.

Aislamiento (Isolation): esta propiedad nos permite que las operaciones en una transacción no afecten otras. Existen diferentes tipos de nivel de aislamiento para las transacciones que podemos utilizar según el contexto y las necesidades propias de la aplicación y/o el flujo de trabajo que desarrollemos.

> La Consistencia nos asegura que la integridad de los datos se mantendrá una vez realizada la transacción y que no se romperá ninguna restricción.

Durabilidad (Durability): esta propiedad nos asegura que una vez realizada la transacción, los datos modificados o agregados quedarán guardados de una manera no-volátil y estarán disponibles aun si ocurriese una falla en el sistema. Esto es cierto aun si la falla ocurre 1 milisegundo después de haber realizado la transacción.

Ejemplo

Ahora veamos cómo implementar transacciones del tipo **Two Phase Commit** en nuestros flujos de trabajo mediante la utilización de Workflow Foundation.

Cabe mencionar que el siguiente ejemplo supone la existencia de la base de datos **Northwind** implementada en SQL Server Express de manera local, y también supone que se ha configurado el servicio **SqlWorkflowPersistenceService** con una base de datos también local llamada **WorkflowStore**, con los scripts que se incluyen en el .NET Framework 3.0 presentes en **[Carpeta de Windows]\Microsoft.NET\Framework\v3.0\Windows Workflow Foundation\ SQL\EN**.

www.reduserspremium.blogspot.com.ar

Primero debemos abrir Visual Studio.NET. Luego crearemos un nuevo proyecto del tipo

La propiedad de Aislamiento nos asegura que las operaciones de una transacción no pueden afectar otras.

Sequential Workflow Console Application. Con el diseñador de WF en la pantalla, arrastremos y coloquemos una actividad del tipo **TransactionScope** tal como lo muestra la Figura 43.

La actividad **TransactionScope** es una actividad compuesta, es decir, que puede contener una actividad o más dentro de ella. Todas las actividades que definamos dentro de la actividad **TransactionScope** participarán en la misma transacción.
Una de las propiedades más importantes de esta actividad es **TransactionOptions.IsolationLevel**, que indica el nivel de aislamiento que esta transacción tendrá al ejecutar todas sus operaciones. Su valor predeterminado es **Serializable** (el nivel de aislamiento más restrictivo). En la Tabla 9 se proporcionar una lista de los diferentes niveles de aislamiento disponibles. No podemos colocar una actividad **TransactionScope** dentro de otra.

Para continuar con el ejemplo usaremos una actividad del tipo **Code** y la colocaremos dentro de la actividad **transactionScopeActivity1**. Es en esta actividad donde declararemos el código que deseamos que participe en la transacción. Por lo tanto, asignemos un método llamado "**Transaccion**" en la propiedad **ExecuteCode** de la actividad **codeActivity1**.

En este método debemos definir el siguiente código:

C#:

```
private void Transaccion(object sender,
EventArgs e) {
   using (SqlConnection conn = new
```

Tabla 9 | Niveles de aislamiento para las transacciones

Nombre	Descripción
Serializable	Los datos no aplicados pueden ser leídos pero no modificados y no se pueden insertar datos durante la transacción.
RepeatableRead	Los datos no aplicados pueden ser leídos pero no modificados. Se pueden insertar datos durante la transacción.
ReadCommitted	Los datos no aplicados no pueden ser leídos durante la transacción pero sí pueden ser modificados.
ReadUncommitted	Los datos no aplicados pueden ser leídos y modificados.
Snapshot	Los datos no aplicados pueden ser leídos. Antes de modificar los datos, es necesario verificar si otra transacción ha modificado los datos después de que éstos se leyeron. Si es así, arroja una excepción.
Chaos	Los cambios pendientes de transacciones que tengan un nivel de aislamiento superior no pueden ser reemplazados.
Unspecified	Otro. Sin embargo, este nivel no puede asignarse de manera manual.

www.reduserspremium.blogspot.com.ar

```
    SqlConnection()) {
      conn.ConnectionString = @"Data Source=
      .\sqlexpress;Initial Catalog=Northwind;
      Integrated Security=Yes";
      SqlCommand cmd = new SqlCommand("INSERT
      INTO Region VALUES (@RegionID,
      @Description)", conn);
      cmd.Parameters.AddWithValue("@RegionID",
      this.RegionID);
      cmd.Parameters.AddWithValue("@Description",
      this.RegionID.ToString());
      conn.Open();
      cmd.ExecuteNonQuery();

    }
}
```

> La Durabilidad nos asegura que una vez realizada la transacción, los datos quedarán guardados aunque falle el sistema.

CompensateActivity o Transacciones Compensables

Este tipo de transacciones nos permiten definir una serie de operaciones a ejecutar cuando la ejecución de la transacción falla. Cuando esto sucede, la compensan. A diferencia de las transacciones del tipo **Two Phase Commit**, donde las operaciones se ejecutan o no se ejecutan, en una transacción compensable las operaciones sí se ejecutan y en caso de una falla esas operaciones no se cancelan sino que se ejecutan otras acciones que complementan o "compensan" esa falla.

Para dar un ejemplo de esto podemos mencionar el caso de cuando pagamos el excedente de algún servicio, donde la operación compensatoria es emitir una nota de crédito o un cupón para hacer uso de ese beneficio posteriormente. Cabe mencionar que este tipo de transacciones son útiles cuando ya no

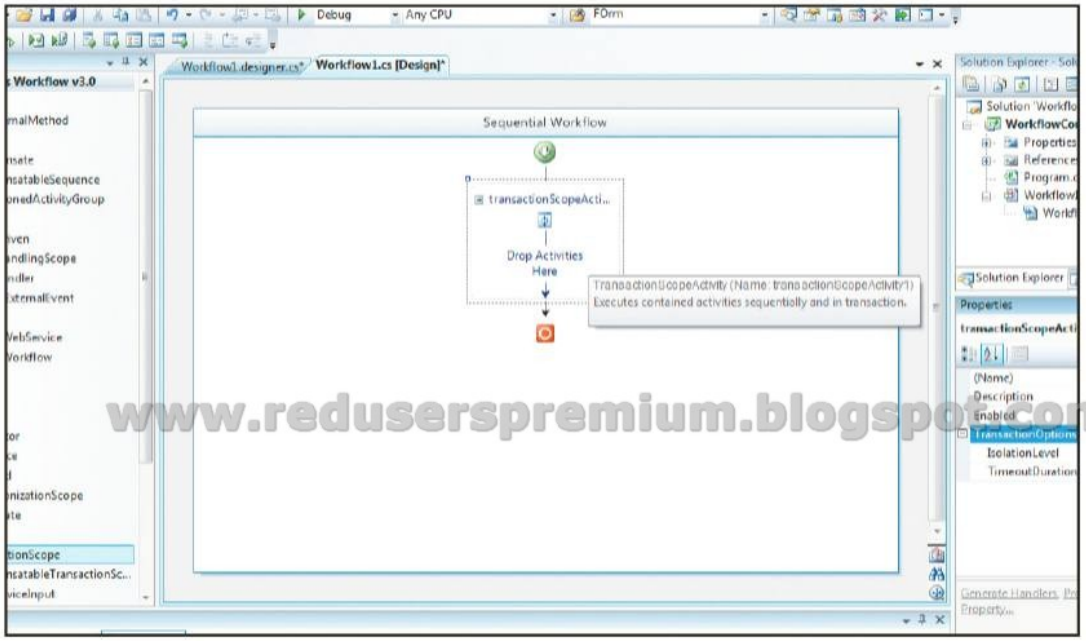


FIGURA 043 | Con VisualStudio y WWF podemos utilizar transacciones de una forma visual y sencilla.

www.reduserspremium.blogspot.com.ar

podemos modificar los cambios producidos por esa transacción o por otra.

Ejemplo

Para demostrar este tipo de transacciones retomemos el proyecto del ejemplo anterior. Sin embargo, en vez de usar una actividad del tipo **TransactionScope** utilizaremos la actividad **CompensatableTransactionScope**, que nos permite definir una serie de actividades a ejecutar como parte de esa transacción, tal como ocurre con su contraparte explicada en la sección anterior. Asimismo, para esta demostración provocaremos un error a propósito después de ejecutar la transacción que da de alta regiones en la base de datos **Northwind**. La actividad que arrojaremos será del tipo **System.Exception**. En la Figura 44 podemos observar el diseñador de Workflow Foundation con los cambios necesarios.

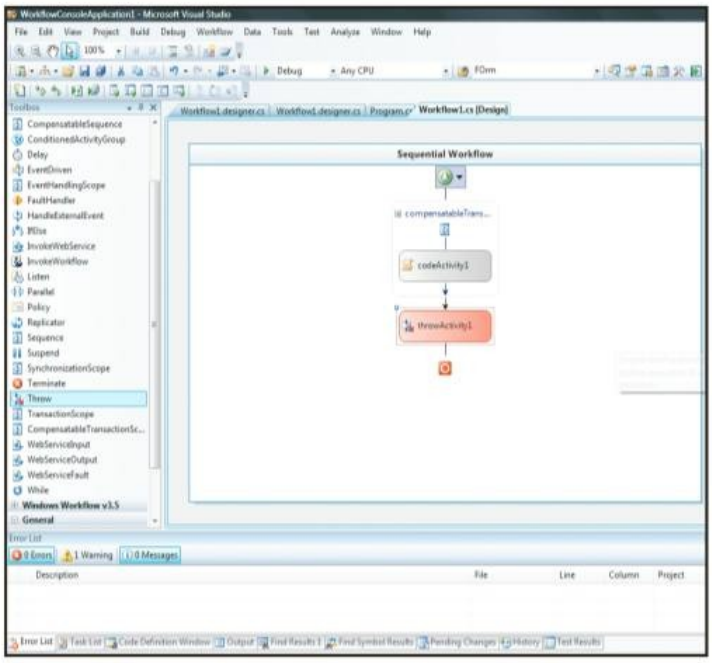


FIGURA 044 | Flujo de trabajo con una actividad del tipo CompensatableTransactionScope.

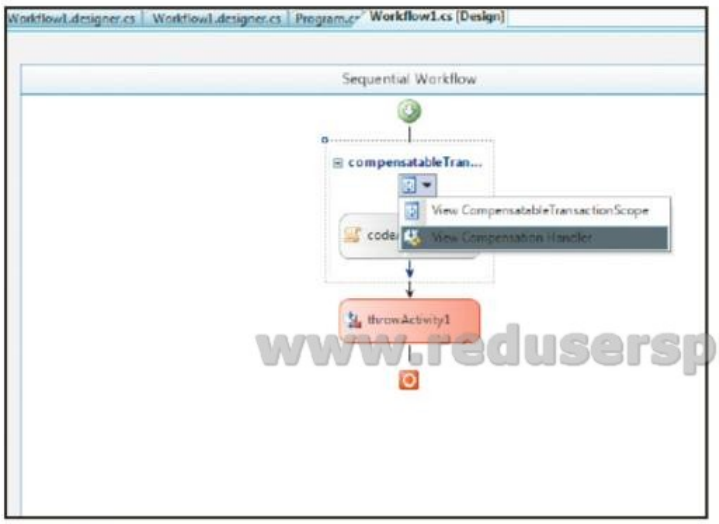


FIGURA 045 | Opciones disponibles en el menú contextual de la actividad CompensatableTransactionScope.

Debemos notar que el tipo de transacción que ejecuta la actividad **codeActivity1** ahora es del tipo **CompensateTransactionScope**. Ahora agregamos una actividad del tipo **Throw** después de la transacción para provocar el error mencionado.

Para indicar la serie de actividades compensatorias a ejecutar debemos definirlas en la vista de la actividad **compensationHandlerActivity1**. Esto lo podemos lograr si hacemos clic sobre la opción apropiada del menú desplegable de la transacción **compensatableTransactionScopeActivity1**, tal como lo muestra la Figura 45.

En la actividad **compensationHandlerActivity1** añadiremos una actividad del tipo **CodeActivity**. Para ello, simplemente arrastramos esta actividad desde la barra de herramientas hasta nuestro flujo de trabajo. Esta actividad nos servirá para ejemplificar la compensación de una transacción cuando ésta falla. A esta actividad le relacionaremos el método Compensa y en este método solamente agregaremos el siguiente código.

www.redusers premium.blogspot.com.ar

C#:

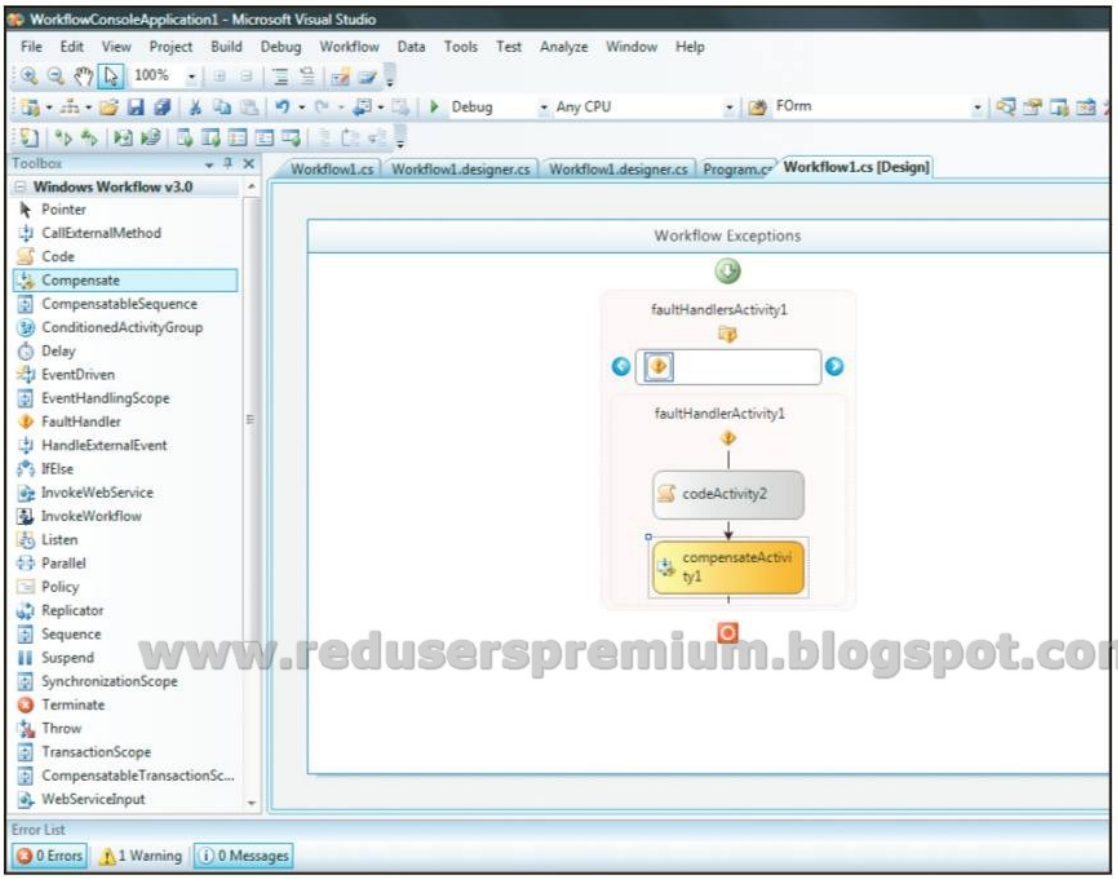
```
...
   private void Compensa(object sender,
   EventArgs e) {
     Console.WriteLine("Compensando...");
   }
...
```

Por último, en la vista de **Fault Handlers** agregaremos una actividad del tipo **Compensate**, que por medio de su propiedad **TargetActivityName** indica la actividad que se quiere compensar. En nuestro caso será la actividad **compensatableTransactionScopeActivity1**. Esta actividad nos servirá para indicar de manera manual que queremos realizar la compensación correspondiente. La Figura 46 muestra el diseñador de Windows Workflow Foundation con la vista de **Fault Handlers**.

> En una transacción compensable, las operaciones que se ejecutan y fallan no se cancelan sino que se ejecutan otras acciones que complementan o "compensan" esa falla.

Ahora estamos en condiciones de ejecutar nuestro proyecto para probar nuestro flujo de trabajo con la actividad de compensación. Recordemos que provocamos un error a propósito y a la vez simulamos la compensación de la transacción.

FIGURA 046 | Vista de las actividades dentro de una actividad del tipo FaultHandler.



www.reduserspremium.blogspot.com.ar

Trabajar con Worflows

Una vez adquiridos los conocimientos técnicos sobre Workflow, es necesario saber cómo trabajar con los flujos de trabajo.

Eventos del Runtime

A continuación profundizaremos un poco más acerca del Runtime Engine, los diferentes eventos del Runtime, su sintaxis y su uso en Workflow Foundation.

Runtime Engine

La arquitectura de Windows Workflow Foundation consta de seis partes principales, el **Runtime Engine** es una de ellas. El **Runtime Engine** es una librería que ejecuta flujos de trabajo. También provee otros servicios, como mecanismos para comunicarse con software fuera del flujo de trabajo.

Todos los flujos de trabajo dependen de Windows Workflow Foundation Runtime Engine. Este motor ejecuta cada flujo de trabajo y maneja su estado durante todo su tiempo de vida. El Windows Workflow Foundation Runtime es una librería. Por lo tanto, debe ser ejecutado en algún proceso host. En lugar de proveer un host requerido, Windows Workflow Foundation le permite al Runtime (y a cualquier flujo de trabajo que ejecute) alojarse en algún proceso de Windows, desde una simple aplicación de consola o una aplicación de formulario, hasta un servidor complejo diseñado teniendo en cuenta el ambiente de flujo de trabajo.

Windows Workflow Foundation cuenta con un conjunto de servicios que le permiten al **Runtime** ejecutarse dentro de **ASP.NET**, aunque los vendedores independientes de software y los usuarios finales son libres de crear sus propias aplicaciones contenedoras de flujo de trabajo para las aplicaciones existentes.

Debido a que Windows Workflow Foundation es un Framework para flujos de trabajo en lugar de un producto independiente, y al soportar este tipo de diversidades, se convierte en una meta explícita para sus creadores. A pesar de que cada aplicación host usa el mismo **Runtime Engine**, cada uno debe proveer un conjunto de **Servicios Runtime**. Estos servicios brindan soporte para persistir el estado de un flujo de trabajo, rastrear su ejecución, usar transacciones, etcétera.

Un flujo de trabajo puede ser de larga duración, ya que podría ejecutarse durante horas, días o semanas, y el Runtime de Windows Workflow Foundation podría apagarse durante un flujo de trabajo, y persistentemente almacenaría su estado si se encontrara inactivo por un período de tiempo.
La decisión de descargar el flujo de trabajo se puede tomar porque éste se encuentra bloqueado a la espera de un evento externo. En general, es el Runtime el que toma esta desición. Para escribir estados del flujo de trabajo en el disco, el Runtime depende del

Windows Workflow Foundation cuenta con un conjunto de servicios que le permiten al Runtime ejecutarse dentro de una aplicación ASP.NET.

www.reduserspremium.blogspot.com.ar

servicio de persistencia proporcionado por su proceso host.

Los eventos del flujo de trabajo son procesados por el Runtime mediante los delegados de manejo de eventos. Los eventos a los que se puede suscribir que se ejecutan del lado del workflowRuntime son:

Del tipo:

- EventHandler<WorkflowRuntime-EventArgs>
- EventHandler<WorkflowEventArgs>

Del motor:

- Started
- Stopped

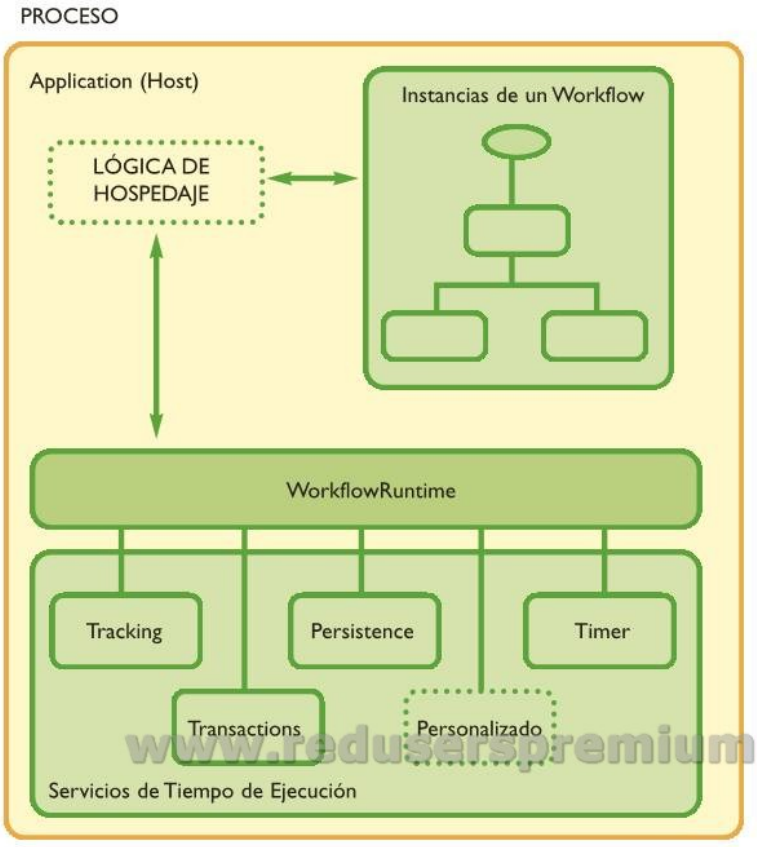
De las instancias:

- WorkflowCreated/Started/Stopped
- WorkflowAborted/Terminated/Completed
- WorkflowIdled/Loaded/Persisted/Unloaded

Eventos de flujo de trabajo

Los eventos del flujo de trabajo son procesados por el Runtime mediante los delegados de

Flujo de trabajo



www.reduserspremium.blogspot.com.ar

FIGURA 047 | Diagrama de interacción del motor de los flujos de trabajo, lógicas e instancias.

manejo de eventos. Estos eventos son generados/levantados por el **Runtime** del flujo de trabajo y no por la instancia del flujo de trabajo. Asimismo, tampoco son modelados como parte de la definición de flujo de trabajo.

Los eventos enviados por el Runtime del flujo de trabajo contienen el identificador de la instancia del flujo de trabajo que generó el evento. La Tabla 10 nos muestra los diferentes eventos de flujo de trabajo existentes.

En el siguiente ejemplo podemos ver la implementación del evento **WorkflowCompleted** del **WorkFlowRuntime**:

C#:

```
...
private void ApplicationInitMethod() {
  WorkflowRuntime workflowRuntime = new
  WorkflowRuntime();
  workflowRuntime.WorkflowCompleted += new
  EventHandler<WorkflowCompletedEventArgs>
  (workflowRuntime_WorkflowCompleted);
}

void workflowRuntime_WorkflowCompleted(object
sender, WorkflowCompletedEventArgs e) {
  MessageBox.Show("Se completó el flujo de
  trabajo");
}
...
```

En el siguiente ejemplo podemos ver la implementación del evento **WorkflowCreated** del **WorkFlowRuntime**:

C#:

```
...
private void ApplicationInitMethod() {
```

Tabla 10 | Eventos de flujo de trabajo

Eventos de flujo de trabajo	Descripción
ServicesExceptionNotHandled	Generado cuando la instancia del flujo de trabajo no puede manejar la excepción interna.
Started	Generado cuando se inicia el Runtime.
Stopped	Generado cuando se detiene el Runtime.
WorkflowAborted	Generado cuando se aborta la instancia del flujo de trabajo.
WorkflowCompleted	Generado cuando se completa la instancia del flujo de trabajo.
WorkflowCreated	Generado cuando se crea la instancia del flujo de trabajo.
WorkflowIdled	Generado cuando la instancia del flujo de trabajo está inactiva (por ejemplo: Delay o EventSink).
WorkflowLoaded	Generado cuando la instancia del flujo de trabajo se carga en la memoria (por ejemplo: Rehydrated).
WorkflowPersisted	Generado cuando la instancia del flujo de trabajo tiene persistencia.
WorkflowResumed	Generado cuando la instancia del flujo de trabajo se reanuda después de ser suspendida.
WorkflowStarted	Generado cuando se inicia la instancia del flujo de trabajo.
WorkflowSuspended	Generado cuando el Runtime suspende la instancia del flujo de trabajo.
WorkflowTerminated	Generado por el Runtime o cuando internamente se termina la instancia del flujo de trabajo.
WorkflowUnloaded	Generado cuando la instancia del flujo de trabajo se descarga de la memoria (por ejemplo: Hydrated).

www.redusserspremium.blogspot.com.ar

```
    WorkflowRuntime workflowRuntime = new
    WorkflowRuntime();
    workflowRuntime.WorkflowCreated += new
    EventHandler<WorkflowCompletedEventArgs>
    (workflowRuntime_WorkflowCreated);
}

void workflowRuntime_WorkflowCreated(object
sender, WorkflowCompletedEventArgs e) {
    MessageBox.Show("Se creó el flujo de
    trabajo");
}
...
```

Métodos del Runtime

En esta sección profundizaremos un poco más acerca de los métodos disponibles en el Runtime, su sintaxis y su uso en Workflow Foundation.

La clase WorkflowRuntime

El Runtime del flujo de trabajo es el intermediario entre la aplicación que hospeda un flujo de trabajo y las instancias del flujo de trabajo. Aunque el flujo de trabajo es una parte muy importante de Windows Workflow Foundation, el runtime juega un papel muy importante en el ciclo de vida de los flujos de trabajo. La clase **System.Workflow.Runtime.WorkflowRuntime** representa el Runtime del flujo de trabajo y proporciona una gran funcionalidad para manejar el ambiente del Runtime. Si utilizamos esta clase, se tiene control absoluto de la ejecución de las instancias del flujo de trabajo y del runtime mismo. La clase **System.Workflow.Runtime.WorkflowRuntime** es responsable de varias tareas que son muy importantes. Una de ellas es la del manejo de los eventos del Runtime que acabamos de ver en la sección anterior. Otra de ellas es la gestión del Runtime.

La clase WorkflowRuntime expone dos métodos públicos que se relacionan con la gestión del Runtime del flujo de trabajo: StartRuntime y StopRuntime.

Gestión del Runtime

El método **StartRuntime** permite que se ejecuten dos acciones importantes. Primero, existen servicios base del Runtime que siempre deben existir en el flujo de trabajo del Runtime: un servicio de transacción del flujo de trabajo y un servicio planificador del flujo de trabajo. Cuando se llama a este método, se realiza una validación para revisar si alguno de estos dos servicios ha sido añadido manualmente al Runtime.

De lo contrario, el Runtime crea instancias por defecto para cada tipo de servicio. La clase por defecto del servicio de transacción es **Default-**

¿Por qué son importantes los Runtime Events?

Porque proporcionan un mecanismo para hacer un seguimiento del ciclo de vida del flujo de trabajo, por ejemplo cuando se crea un flujo de trabajo, cuando finaliza el mismo flujo, etcétera. Otro beneficio de manejar eventos del Runtime es que proporcionan un mecanismo para hacer un seguimiento de todas las instancias del flujo de trabajo. De esta forma, podemos determinar cuántos flujos de trabajo fueron creados, abortados, terminados, etcétera.

www.redusersporemium.blogspot.com.ar

WorkflowTransactionService, y la clase por defecto del servicio de planificación es **DefaultWorkflowSchedulerService**. Después de que los servicios se agregaron y se instanciaron exitosamente al Runtime, cada servicio se inicia con el método **Start**. Además de la configuración de los servicios que se produce durante el proceso de arranque del Runtime, a la propiedad **IsStarted** del Runtime se le otorga el valor **true**, y se genera el evento **Started**.

Llamar al método **StopRuntime** tiene el efecto contrario. Se detienen todos los servicios, se descargan todas las instancias de flujos de trabajo, a la propiedad **IsStarted** se le otorga el valor **false**, y se genera el evento **Stopped**. La Tabla 11 nos muestra los diferentes métodos de trabajo existentes en el workflow Runtime.

Iniciar y manejar instancias del Workflow

Una de las tareas más importantes que el Runtime puede realizar es iniciar instancias del flujo de trabajo. Además de iniciar las instancias, el Runtime expone la funcionalidad para manejarlos. Para manejar una instancia del flujo de trabajo, simplemente hay que llamar el método **CreateWorkflow** de la instancia **WorkflowRuntime**. Existen muchas formas de llamar a este método, pero el más usado es el que usa una instancia **Type**, que representa el tipo de la clase de flujo de trabajo. Por ejemplo:

C#:

```
...
//- MiWorkflow es la definición de clase del
workflow
Type workflowType = typeof(MiWorkflow);
//- usa el runtime del workflow runtime para
crear una instancia de MiWorkflow
WorkflowInstance workflowInstance =
    theRuntimeInstance.CreateWorkflow
    (workflowType);
...
```

Aunque el código anterior crea una instancia del flujo de trabajo, en realidad no inicia el

Tabla 11 | Métodos del Workflow Runtime

Método	Descripción
AddService	Le agrega un servicio específico al motor de tiempo de ejecución.
Dispose	Libra los recursos utilizados por el objeto WorkflowRuntime.
CreateWorkflow	Crea una instancia del flujo de trabajo mediante el uso de los parámetros especificados.
GetAllServices	Regresa todos los servicios que han sido agregados al Engine del Runtime del flujo de trabajo que derivan del tipo especificado.
GetLoadedWorkflows	Obtiene la colección que contiene todas las instancias de flujos de trabajo actualmente cargadas en la memoria.
GetService	Obtiene el servicio del Engine del Runtime del flujo de trabajo que usa el tipo especificado.
GetWorkflow	Obtiene las instancia de flujo de trabajo que tienen el Guid especificado.
RemoveService	Quita el servicio especificado del Engine del Runtime del flujo de trabajo.
StartRuntime	Inicia el Engine del Runtime del flujo de trabajo y los servicios del Engine del Runtime del flujo de trabajo.
StopRuntime	Detiene el Engine del Runtime del flujo de trabajo y los servicios del Engine del Runtime del flujo de trabajo.

www.redusersstoremium.blogspot.com.ar

flujo de trabajo. El método **Start** de la clase **WorkflowInstance** es el que lo hace, como se puede ver en el siguiente código:

C#:

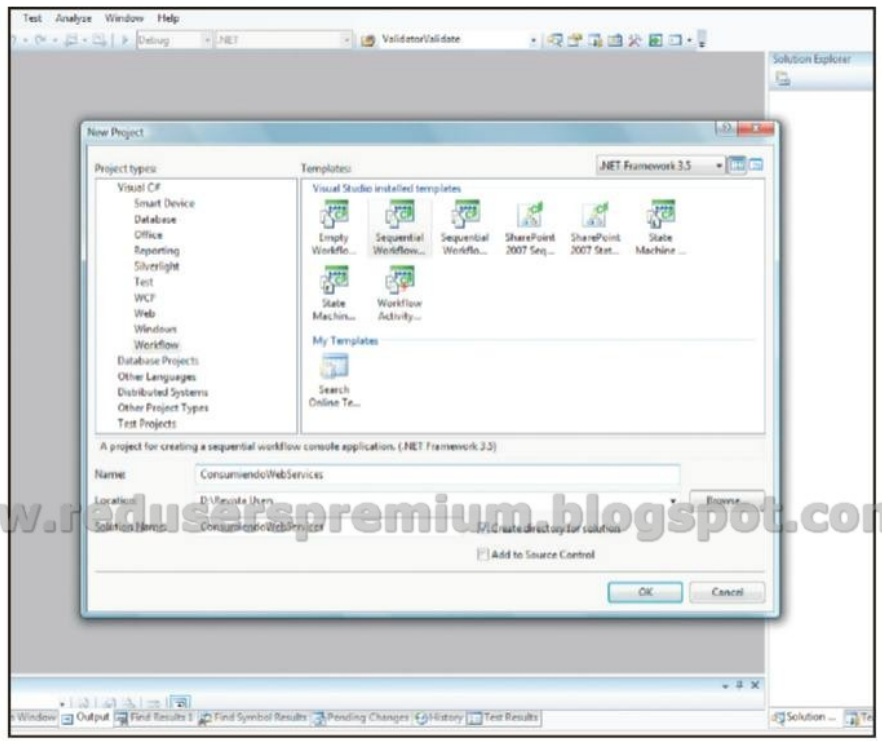
```
...
//- Inicia la instancia del flujo de trabajo
workflowInstance.Start();
...
```

En el siguiente ejemplo podemos ver la ejecución de los métodos **StartRuntime** y **StopRuntime** del **WorkFlowRuntime** junto con el método **Start** para la instancia del flujo de trabajo:

C#:

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

      using (WorkflowRuntime workflowRuntime
      = new WorkflowRuntime()) {
        ExternalDataExchangeService
        dataService = new ExternalData
        ExchangeService();
        workflowRuntime.AddService
        (dataService);
        dataService.AddService
        (expenseService);

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

        workflowRuntime.WorkflowCompleted +=
        OnWorkflowCompleted;
        workflowRuntime.WorkflowTerminated +=
        OnWorkflowTerminated;

        Type type = typeof(EjemploWorkflow1);
        WorkflowInstance workflowInstance =
```

FIGURA 048 | Cuadro de diálogo donde podemos escoger el tipo de proyecto a crear.



www.readersspremium.blogspot.com.ar

```
            workflowRuntime.CreateWorkflow(type);
            workflowInstance.Start();

            waitHandle.WaitOne();

            workflowRuntime.StopRuntime();
        }
    }
...
```

Consumo de Web Services

Hasta este momento hemos visto y explicado una gran cantidad de funcionalidades de Workflow Foundation implementadas en varias actividades que vienen incluidas. En este capítulo explicaremos una más: la actividad **InvokeWebService**, que nos permite realizar la invocación a un servicio web XML y de esa manera extender la funcionalidad de nuestros flujos de trabajo.

Si bien está fuera del alcance de este capítulo explicar en detalle el funcionamiento de los servicios web, debemos recordar que son componentes implementados en servidores web y que exponen una serie de funcionalidades que podemos utilizar y volver a utilizar en las aplicaciones que desarrollemos.

Para demostrar lo fácil que es consumir servicios web desde nuestros flujos de trabajo en Workflow Foundation, comencemos por Visual Studio.NET y creemos una nueva **aplicación de consola de flujo de trabajo secuencial**. Para esto hay que seleccionar la plantilla "**Sequential Workflow Console Application**" en la ventana **New Project** de Visual Studio.NET. A esta solución le pondremos el nombre de **ConsumodeWebServices** tal como lo muestra la Figura 48.

A nuestro proyecto le agregaremos un nuevo sitio web del tipo **ASP.NET Web Service** y le pondremos el nombre de **WSPrueba**. Éste

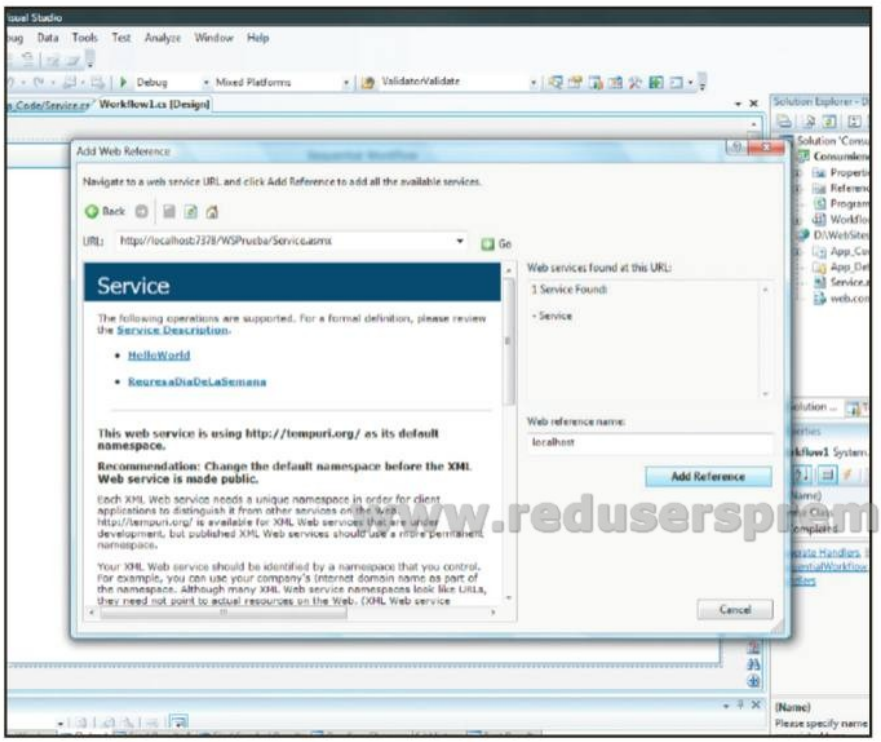


FIGURA 049 | VisualStudio.NET nos ofrece una manera sencilla de agregar referencias web.

www.reduserspremium.blogspot.com.ar

será el sitio donde desarrollaremos un servicio web de prueba para demostrar la invocación desde un flujo de trabajo. Cuando creamos un proyecto de este tipo, automáticamente Visual Studio.NET agrega un servicio web llamado **Service.asmx** y su archivo de code-behind relacionado **Service.asmx.cs**. Nuestro servicio web tendrá un método público que regresará el día de la semana actual. Para esto necesitamos implementar el siguiente código:

C#:

```
...
  [WebMethod]
  public string RegresaDiaDeLaSemana() {
    return DateTime.Now.DayOfWeek.ToString();
  }
...
```

Recordemos que aquellos métodos que deseamos invocar mediante el servicio web deberán estar decorados con el atributo **WebMethod** y tener el modificador de acceso **public**.

Ahora bien, en el diseñador de nuestro flujo de trabajo agreguemos una actividad del tipo **InvokeWebService**. Al hacer esto, Visual Studio.NET automáticamente nos presenta el cuadro de diálogo **Add Web Reference** (el mismo que se muestra al agregar una referencia web en cualquier tipo de proyecto). El objetivo de este cuadro de diálogo es indicar cuál es el servicio web que deseamos invocar por medio de la actividad **InvokeWebService**. En nuestro caso seleccionaremos "**Web Services in this solution**" ya que esta opción buscará los servicios web implementados en la misma solución.

Una vez seleccionada esta opción, el cuadro de diálogo nos muestra una lista con los servicios web encontrados y luego seleccionaremos el servicio llamado "**Service**", que es el que creamos en los párrafos anteriores. Al hacer clic en este servicio, el cuadro de diálogo nos muestra su detalle completo (ver Figura 49).

Hagamos clic en el botón **Add Reference**. Como se espera, Visual Studio.NET agrega al proyecto de nuestro flujo de trabajo la referencia web correspondiente al servicio seleccionado. Además, la actividad **InvokeWebService** se muestra en el diseñador. No obstante, hasta este momento aún no hemos configurado esa actividad, lo que requiere establecer la propiedad **MethodName** que indica el método del servicio web que se ejecutará. En la ventana de propiedades **MethodName** se muestra una lista desplegable. Al hacer clic sobre ella muestra la lista de todos los métodos disponibles en el servicio.

En nuestro caso seleccionaremos **RegresaDiaDeLaSemana**. Al seleccionar dicho método, la ventana de propiedades se actualiza y muestra una nueva propiedad llamada (**ReturnValue**). Ésta nos permite mapear el valor de retorno del método a alguna propiedad o a algún campo dentro de nuestro flujo de trabajo. Esto lo podemos lograr al hacer clic sobre

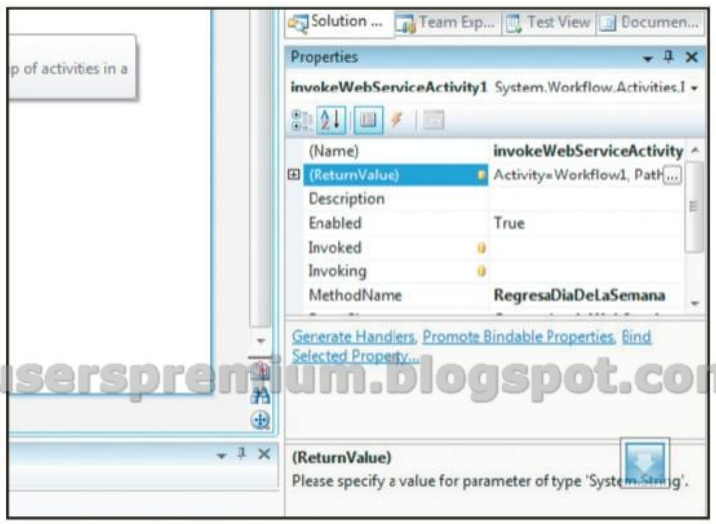


FIGURA 050 | Ejemplo de cómo debe quedar configurada la propiedad InvokeWebService.

www.reduserspremium.blogspot.com.ar

el botón de tres puntos "…" en la propiedad (**ReturnValue**) y configurar el mapeo, en nuestro caso, a un campo llamado "resultado".

Muy bien, ahora para probar la invocación del servicio web agreguémosle una actividad de tipo **Code** al diseñador del flujo de trabajo. A esta actividad le relacionaremos el método **Consulta** en su propiedad **ExecuteCode** tal como lo muestra la Figura 51.
El mismo mecanismo de asignación se usa cuando el método web requiere parámetros (ver Figura 50).

El método Consulta tendrá el siguiente código, que muestra el resultado de la invocación de nuestro servicio web:

C#:

```
private void Consulta(object sender,
EventArgs e) {
    Console.WriteLine(resultado);
}
```

Hemos finalizado. Para probar el trabajo ejecutemos nuestra aplicación. Para esto debemos presionar CTRL + **F5** o debemos hacer clic en el menú **Debug/Start Without Debugging**. Dependiendo del día en el que se ejecute el código, el resultado nos mostrará el día correcto de la semana (Figura 52).

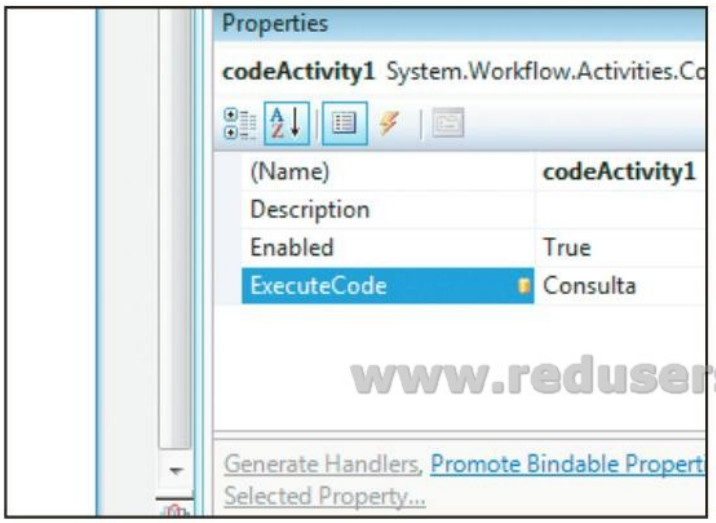


FIGURA 051 | Ejemplo de cómo debe quedar configurada la propiedad ExecuteCode.

Publicación de flujos de trabajo como servicios web

Una de las características más novedosas y útiles que podemos encontrar al desarrollar flujos de trabajo con Workflow Foundation es su capacidad de exponer nuestros flujos como servicios web y usarlos en cualquier tipo de aplicaciones. A continuación veremos qué se requiere para realizar esta funcionalidad y también explicaremos cómo invocar nuestros flujos de trabajo como servicios web desde una aplicación de consola.
Para crear un flujo de trabajo que sirva como servicio web, vamos a crear un nuevo proyecto del tipo **Console Application**, y le daremos el nombre de **AplicacionCliente**. Esta aplicación nos servirá para invocar el servicio web que expondrá la funcionalidad del flujo de trabajo.
A la solución le agregaremos un nuevo proyecto de tipo **Sequential Workflow Library**. Para esto debemos hacer clic con el botón derecho del mouse sobre la solución en **Solution Explorer** y seleccionar la opción **Add/New Project**. En el cuadro de diálogo seleccionaremos la plantilla **Sequential Workflow Library**, y asignaremos "**WorkflowEjemplo**" como nombre.
Ahora bien, necesitamos definir una interfaz que nos servirá para establecer el conjunto de métodos que nuestro servicio web expondrá. Para realizar esto, hagamos clic con el botón derecho del mouse sobre el proyecto **WorkflowEjemplo** en **Solution Explorer** y seleccionemos la opción **Add/New Item**

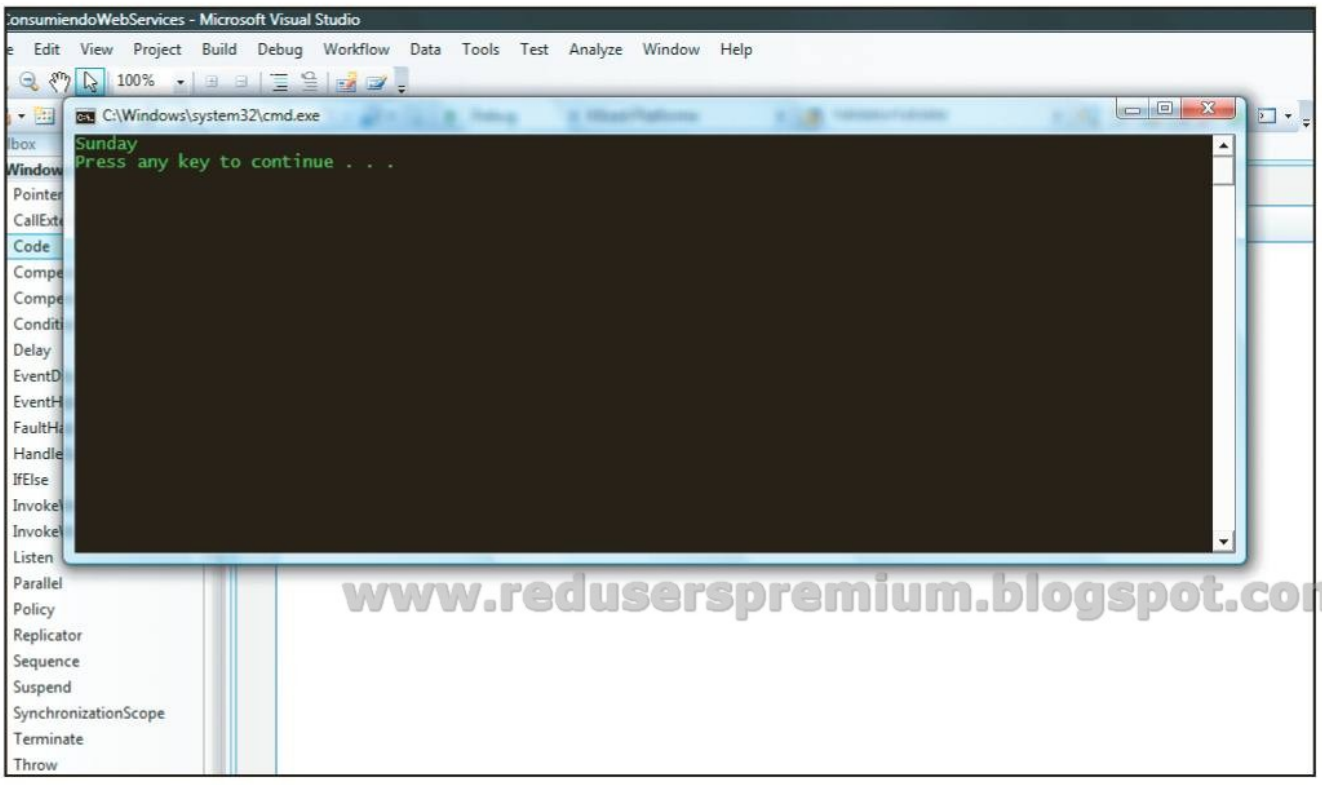
www.reduserspremium.blogspot.com.ar

del menú. Al hacer esto, Visual Studio.NET despliega el cuadro de diálogo Add New Item con una serie de plantillas a escoger y una caja de texto para asignar el nombre del nuevo elemento.
Seleccionemos la plantilla **Interface** del panel derecho y asignemos "**IEjemplo.cs**" como nombre de la nueva interfaz. Al hacer clic en Ok, Visual Studio.NET nos muestra el diseñador de código de la interfaz agregada. Asignemos el siguiente código en IEjemplo.cs:

C#:

```
public interface IEjemplo {
    string RegresaSignoZodiacal(DateTime
    fechaNacimiento);
}
```

Una vez definida la interfaz para nuestro servicio web, regresemos al diseñador del flujo de trabajo. Luego arrastremos y coloquemos una actividad del tipo **WebServiceInput**, que le indica al flujo de trabajo que obtendrá una solicitud externa mediante una aplicación, es decir, que será invocado como un servicio web.
La actividad **WebServiceInput** debe ser configurada. Su propiedad **Interface Type** es la más importante. Esta propiedad indica el tipo de interfaz relacionada con esta actividad y para este ejemplo usaremos la interfaz **IEjemplo** creada en los párrafos anteriores. Para asignarle la interfaz **IEjemplo** a la propiedad **InterfaceType** hacemos clic sobre el botón con tres puntos "…" en la ventana de propiedades de Visual Studio.NET. Al hacer clic sobre ese botón se despliega un cuadro de diálogo como muestra la Figura 53.
La siguiente propiedad que debemos configurar es **MethodName**, que como podemos deducir será **RegresaSignoZodiacal**. Al seleccionar este método de la lista desplegable, automáticamente la ventana de propiedades se



www.reduserspremium.blogspot.com.ar

FIGURA 052 | Resultado del consumo de un WebService en nuestro flujo de trabajo.

VisualStudio.NET nos ofrece una forma sencilla de publicar un flujo de trabajo como un servicio web. Sólo tenemos que elegir la opción Publish as WebService en el menú contextual de la solución.

actualiza para mostrarnos **fechaNacimiento**. Esta nueva propiedad la debemos mapear a un campo o a una propiedad en nuestro flujo de trabajo para tener acceso al dato pasado como parámetro cuando se invoque este servicio web. Seleccionamos la propiedad **fechaNacimiento** y hacemos clic en la opción "**Bind Property 'fechaNacimiento'**" en el panel de opciones de la parte inferior de la ventana de propiedades. En el cuadro de diálogo generamos un nuevo miembro del tipo campo llamado **fechaNacimiento** (ver Figura 54).

La última propiedad que configuraremos de la actividad **WebServiceInput** será **IsActivating**, que indica si esta actividad activará el flujo de trabajo. Ahora agreguemos una actividad del tipo **IfElse** a nuestro diseñador, y como expresión condicional en la rama izquierda asignemos un **Code Condition** llamado **EsFechaValida**. Este método validará que la fecha que asigne el usuario como parámetro no sea después de la fecha actual. Entonces, el código para este método será el siguiente:

C#:

```
private void EsFechaValida(object sender,
ConditionalEventArgs e) {
    e.Result = fechaNacimiento<=DateTime.
    Now.Date ? true : false;
}
```

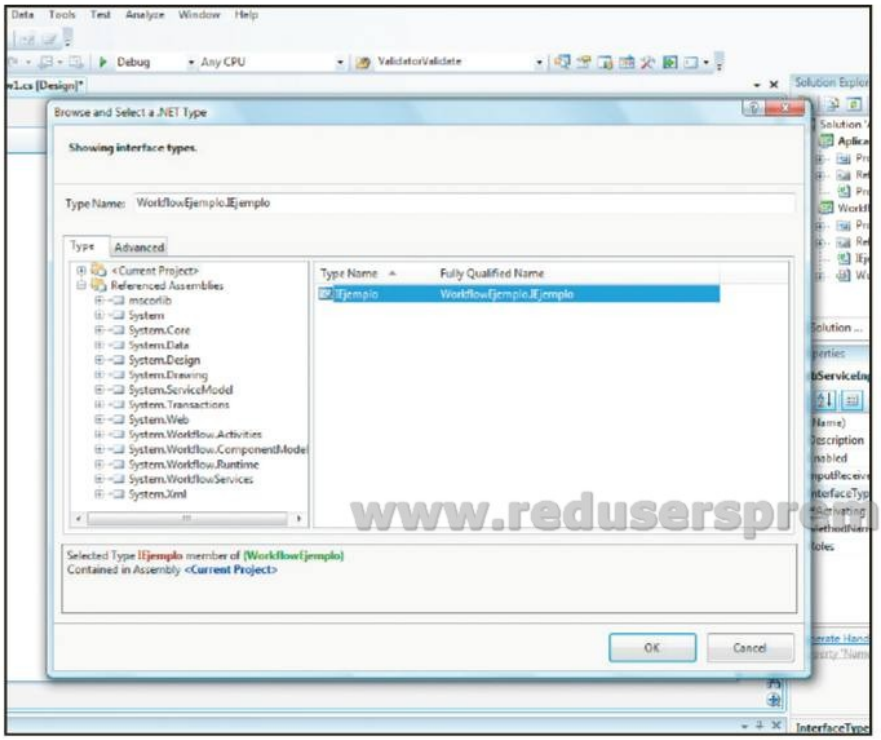


FIGURA 053 | Configurar la propiedad InterfaceType es sencillo si usamos los asistentes de VisualStudio.NET.

www.reduserspremium.blogspot.com.ar

Como paso siguiente agregamos una actividad del tipo **Code** en la rama izquierda de la actividad **IfElse** y una actividad del tipo **WebServiceOutput**. A la actividad **Code** le asignamos el método **CalculaSigno** en su propiedad **ExecuteCode**, y a la actividad **WebServiceOutput** le asignamos el nombre de nuestra actividad **WebServiceInput** (**webServiceInputActivity1**) en su propiedad **InputActivityName**. Al hacer esto, la ventana de propiedades se actualiza para mostrarnos una nueva propiedad llamada **ReturnValue**. Esta propiedad la mapearemos a un nuevo miembro del tipo **field** llamado resultado.

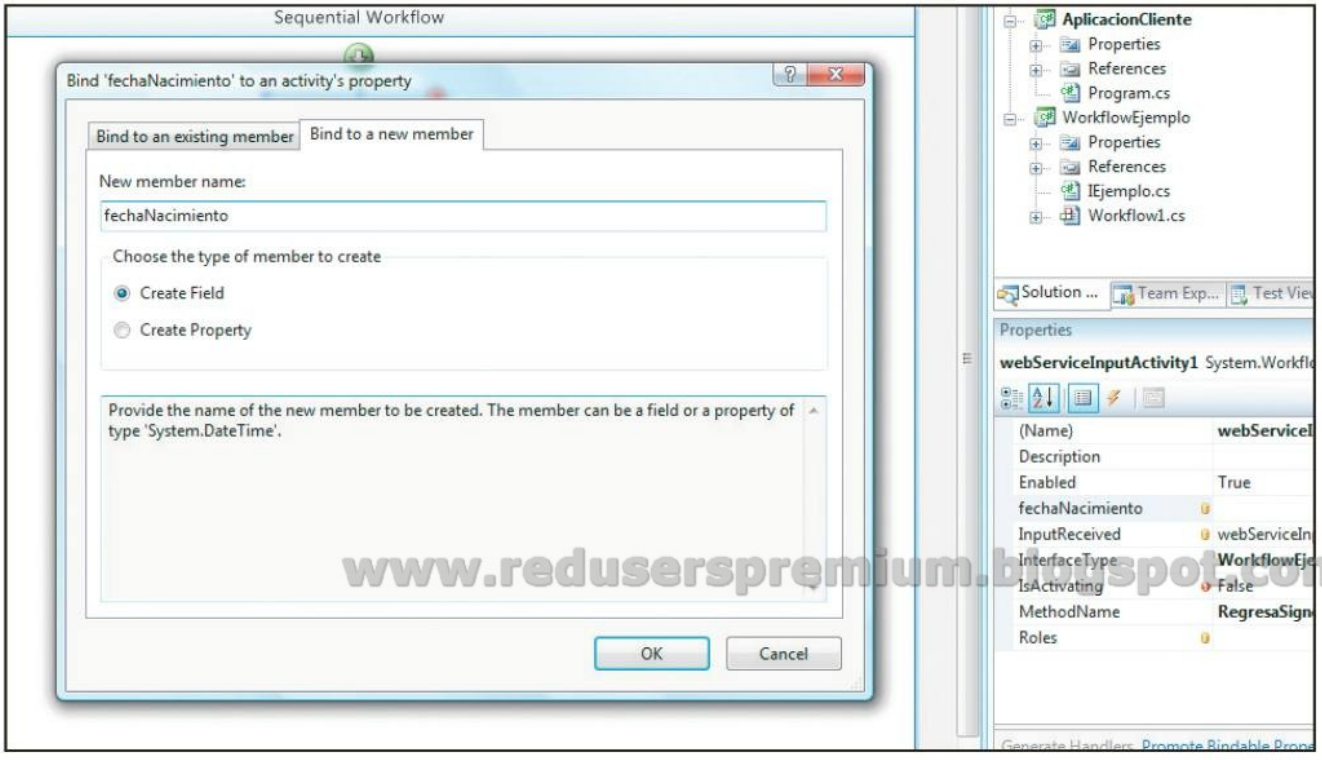
En la rama derecha de la actividad **IfElse** arrastramos y colocamos una actividad del tipo **WebServiceFault** con su propiedad **InputActivityName** mapeada a **webServiceInputActivity1**, y su propiedad **fault** a un nuevo miembro del tipo **field** llamado **fault**.

El siguiente código muestra el método CalculaSigno, que contiene una implementación muy sencilla para calcular el signo zodiacal según la fecha especificada en el parámetro:

> La última propiedad que configuraremos de la actividad WebServiceInput será la IsActivating, que indica si esta actividad activará el flujo de trabajo.

C#:

```
private void CalculaSigno(object sender,
EventArgs e) {
   switch (fechaNacimiento.Month) {
     case 1:
       resultado = fechaNacimiento.Day <=
       21 ? "Capricornio" : "Acuario";
```



www.reduserspremium.blogspot.com.ar

FIGURA 054 | Ejemplo de cómo vincular propiedades a un flujo de trabajo.

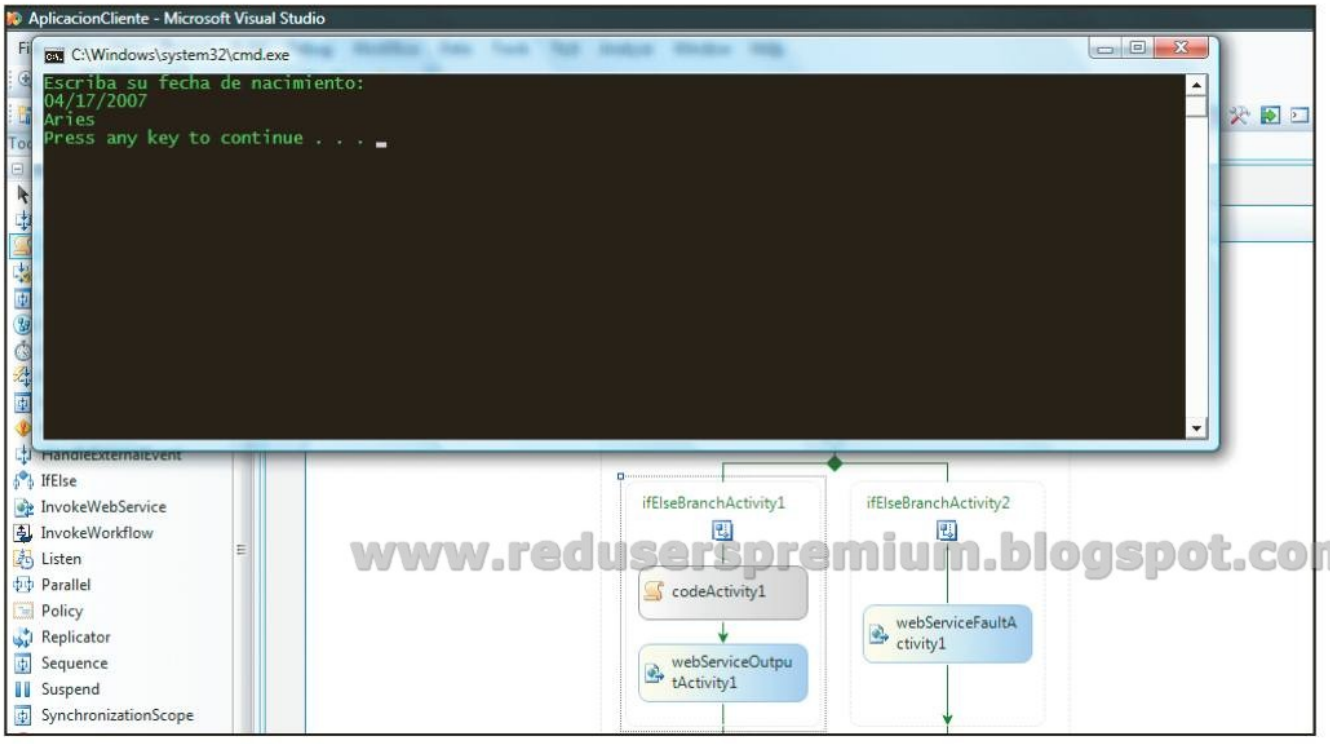
```
        break;
      case 2:
        resultado = fechaNacimiento.Day <=
        19 ? "Acuario" : "Piscis";
        break;

      //Otros meses…
    }
}
```

Para exponer nuestro flujo de trabajo como servicio web, simplemente debemos hacer clic con el botón derecho de mouse sobre el proyecto del flujo de trabajo y seleccionar la opción "**Publish as Web Service**". Al hacer esto, Visual Studio.NET creará automáticamente un proyecto web en nuestra solución, que contiene el servicio web que expone la funcionalidad del flujo de trabajo.
Por último, en la aplicación de la consola agregamos la referencia web que apunta al proyecto que se creó en el párrafo anterior y en el método **Main**() escribimos el siguiente código:

C#:

```
static void Main(string[] args) {
  Console.WriteLine("Escriba su fecha de
  nacimiento: ");
  DateTime fechaNacimiento = Convert.ToDate
  Time(Console.ReadLine());

  localhost.Workflow1_WebService ws = new
  AplicacionCliente.localhost.Workflow1_
  WebService();

  Console.WriteLine(ws.RegresaSignoZodiacal
  (fechaNacimiento));
}
```

Al ejecutar nuestra aplicación y escribir una fecha válida (que sea igual o anterior a la fecha actual), nos mostrará el signo zodiacal relacionado con la fecha especificada.



www.reduserspremium.blogspot.com.ar

FIGURA 055 | Ejemplo de la ejecución de la aplicación que consume el flujo de trabajo publicado como servicio web.

USERS

CURSOS INTENSIVOS

CURSOS.REDUSERS.COM

SALIDA LABORAL

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

24

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Creación de Workflows

Tipos de flujos de trabajo / Reglas y condiciones / Políticas / Tipos de actividades predefinidas

www.redusers-premium.blogspot.com.ar

ISBN 978-987-1347-43-8
00024
9 789871 347438