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

22

Windows Communication Foundation

Service, Data y Message Contracts / Binding / MSMQ: Conceptos e integración / Web Service Enhancements / Remoting

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ISBN 978-987-1347-43-8
00022
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!

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

Web Services Enhancements (WSE)

Microsoft Web Services Enhancements (WSE) 3.0 brinda una plataforma para implementar algunos de los estándares WS-*.

Organizaciones como el *World Wide Web Consortium* (W3C) y la *Organization for the Advancement of Structured Information Standards* (OASIS) han desarrollado estándares y especificaciones para asegurar la interoperabilidad, seguridad y confiabilidad de los servicios Web a través de diversas plataformas tecnológicas.
Algunas especificaciones, como WS-Security 1.0 y 1.1, WS-Trust, WS-SecureConversation, WS-Addressing y Message Transmission Optimization Mechanism (MTOM), fueron creadas en colaboración con Microsoft y otras industrias. Debido a que muchos de sus nombres comienzan con "WS-", se las conoce como WS-*.

Web Services Enhancements (WSE) para Microsoft .NET es un add-on para Microsoft Visual Studio y Microsoft .NET Framework que permite a los desarrolladores construir servicios Web basados en los estándares WS-*. El foco principal de WSE está puesto en brindar seguridad a nivel de mensaje en el desarrollo de aplicaciones distribuidas. Debemos asegurar la autenticidad y confidencialidad de los mensajes manejados por nuestros servicios Web. Las soluciones de seguridad a nivel de protocolo de transporte, como SSL (punto a punto), presentan limitaciones si los mensajes deben ser examinados o necesitan ser procesados por algún intermediario (por ejemplo, no puede encriptarse sólo una parte de ellos). La especificación WS-Security establece cómo podemos crear servicios Web seguros a nivel de mensaje (final a final). WS-Addressing provee la forma de especificar la información de destino y ruteo dentro de los mensajes SOAP, para hacerlos independientes del protocolo de transporte.

WSE también nos permite el envío eficiente de datos binarios (documentos e imágenes) como parte de un mensaje SOAP. Esto se hace a través de un proceso de optimización (transparente para el desarrollador) provisto por MTOM. Otro beneficio es que MTOM permite securitizar tanto los datos como el mensaje SOAP usando WS-Security. MTOM es una recomendación de W3C como reemplazo de DIME y WS-Attachments en el envío de gran cantidad de datos.

Las dos últimas versiones provistas por Microsoft como descargas son WSE 2.0 SP3 y WSE 3.0. El primero es soportado por el .NET Framework 1.1 y 2.0. El segundo está

Web Services Enhancements (WSE) para Microsoft .NET es un add-on para Visual Studio .NET que permite a los desarrolladores construir servicios Web basados en los estándares WS-*.

construido para los desarrolladores que usan Visual Studio 2005 y .NET Framework 2.0. Esta versión nos otorga una herramienta de tiempo de diseño (WSE Settings 3.0) que se integra con Visual Studio 2005. WSE 3.0 y WSE 2.0 SP3 pueden ser instalados lado a lado en la misma máquina. Pero WSE 2.0 no es "wire-level" compatible con WSE 3.0 (ni con WCF), debido a cambios en las especificaciones (como WS-Addressing). Por su parte, WSE 3.0 es "wire-level" compatible con WCF. Esto significa que si desarrollamos e instalamos servicios Web con WSE 3.0, los clientes WCF pueden comunicarse con ellos. De la misma forma, los clientes WSE 3.0 pueden comunicarse con servicios WCF.

El Basic Profile es definido por la organización *Web Service Interoperability* (WS-I), la cual promueve la interoperabilidad de servicios Web entre plataformas, sistemas operativos y lenguajes de programación. Es una guía para la construcción de servicios Web que pueden ser consumidos a través de distintas plataformas tecnológicas. WCF conforma el Basic Profile 1.1. Debemos asegurarnos de que nuestros servicios Web conformen el Basic Profile 1.1. En Visual Studio 2005, el template para la creación de servicios Web hace al servicio compatible con él usando el atributo WebServiceBinding para decorar el servicio. A continuación, veremos unos ejemplos en VB.NET y C#.NET:

VB:

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

''' <summary>
''' Breve descripción para MiWebService
''' </summary>
<WebService(Namespace:="http://tempuri.org/")>
<WebServiceBinding(ConformsTo:=WsiProfiles.
BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.
DesignerGenerated()> _
Public Class MiWebService
  Inherits System.Web.Services.WebService

  <WebMethod()> _
```

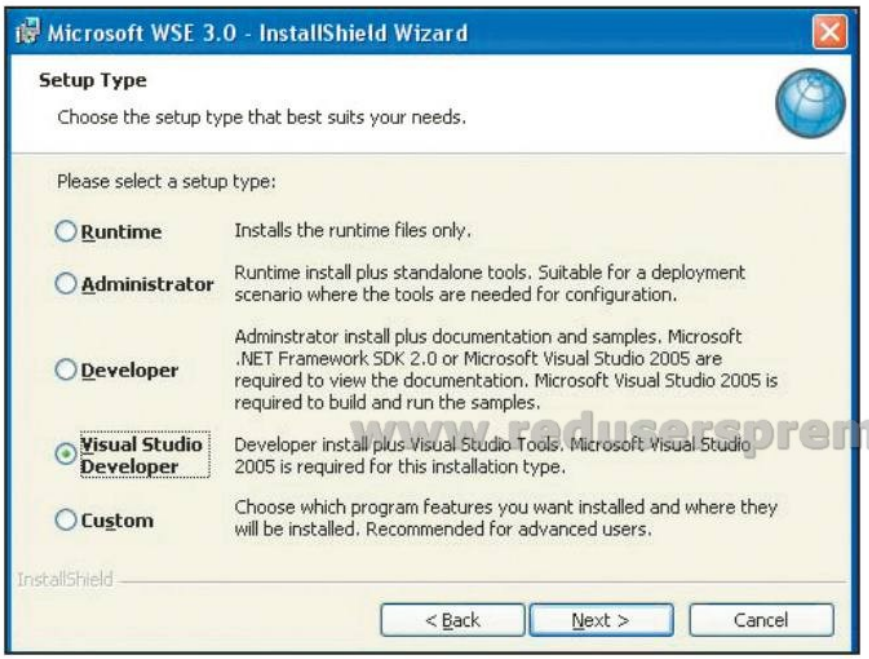


FIGURA 010 | Pantalla de instalación de WSE 3.0.

```
    Public Function HelloWorld() As String
        Return "Hello World"
    End Function

End Class
```

C#:

```
using System;
using System.Collections;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

/// <summary>
/// Breve descripción para MiWebService
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.
BasicProfile1_1)]
public class MiWebService : System.Web.
Services.WebService {

    public MiWebService() {

        //Uncomment the following line if using
        designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }

}
```

WSE también nos permite el envío eficiente de datos binarios como parte de un mensaje SOAP. Esto se hace a través de un proceso de optimización provisto por MTOM.

El gran beneficio de WSE 3.0 es que introduce un framework fácil de usar para otorgar seguridad y autenticación a los servicios Web, abstrayendo a los desarrolladores de los detalles de las especificaciones soportadas (ver Tabla 18). Cada versión de una especificación posee un namespace XML.

Tabla 18 | Especificaciones soportadas por WSE 3.0

Área	Especificación
Seguridad	Web Services Security: SOAP Message Security (WS-Security) 1.0 and 1.1
	Web Services Secure Conversation Language (WS-SecureConversation)
	Web Services Trust Language (WS-Trust)
	Web Services Security X.509 Certificate Token Profile
	Web Services Security Username Token Profile 1.0
	Web Services Security Kerberos Token Profile 1.0
Mensajes	Web Services Addressing (WS-Addressing)
	SOAP Message Transmission Optimization Mechanism (MTOM)
	SOAP 1.1
	SOAP 1.2

.NET Remoting

Remoting permite la comunicación entre objetos en diferentes dominios de aplicación usando distintos protocolos.

Podemos usar Remoting para resolver los siguientes tipos de comunicación entre objetos en diferentes dominios de aplicación: en el mismo proceso, en procesos distintos en la misma PC, y en procesos distintos en computadoras diferentes conectadas por una red.

Remoting usa el modelo cliente-servidor. Los clientes y servidores que emplean Remoting pueden ejecutarse en cualquier entorno de ejecución provisto por el Framework .NET, por ejemplo, en aplicaciones de consola. Un objeto considerado remoto puede ser accedido fuera de su dominio de aplicación por medio de un proxy, o puede ser copiado y su copia pasada fuera de su dominio de aplicación. Esto es, los objetos remotos pueden ser pasados por referencia o por valor. Los primeros se denominan objetos de tipo **Marshal-by-reference** (MBR) y los segundos, de tipo **Marshal-by-value** (MBV). Recordemos que marshalling es el proceso de transformar la representación en memoria de un objeto en un formato que pueda ser usado para persistencia o transmisión a través de una red. La manera más sencilla de definir una clase en Visual Basic que nos permita obtener instancias de tipo MBV es usar la clase **System.SerializableAttribute**:

```
<Serializable()> _
Public Class Test
End Class
```

.NET Remoting

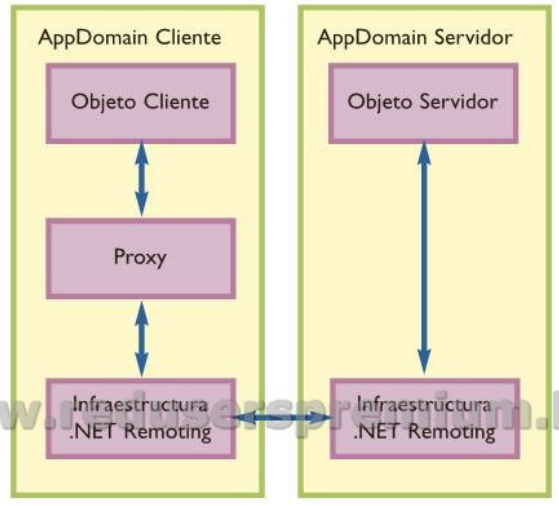


FIGURA 011 | Esquema de comunicación.

Si la clase necesita controlar su proceso de serialización, puede implementar la interfaz **ISerializable**:

```
<Serializable()> _
Public Class Test
    Implements ISerializable
End Class
```

Obtendremos instancias de tipo MBR si la clase deriva de **System.MarshalByRefObject**:

```
Public Class Test
    Inherits MarshalByRefObject
End Class
```

Dependiendo del tipo de activación declarado, cuando un cliente crea una instancia de un objeto MBR, la infraestructura de .NET Remoting genera un proxy que representa al objeto MBR en el dominio de aplicación del cliente, y a dicho cliente le retorna una referencia al objeto proxy.

Cuando el cliente hace una llamada a un método, lo hace sobre el proxy. La infraestructura de .NET Remoting recibe la llamada, hace el marshalling correspondiente hacia el dominio de aplicación del objeto remoto e invoca la llamada sobre el objeto remoto. Finalmente, retorna el resultado al proxy, que lo devuelve al cliente.

Para establecer comunicación entre un cliente y un servidor, debemos configurar, mediante código o usando archivos de configuración, el canal (channel), el formateador (formatter) y el modo de activación.

Los canales (*channels*) son los objetos que transportan mensajes entre dominios de aplicaciones, procesos y computadoras. Definen el protocolo de comunicación y los end-points que el cliente

Tabla 19 | Canales implementados en .NET Framework 2.0

Clase	Namespace	Descripción
HttpChannel	Http	Implementa un canal cliente y servidor (interfaces IChannelReceiver e IChannelSender) que usa HTTP para transmitir mensajes.
HttpClientChannel	Http	Implementa un canal cliente (interfaz IChannelSender) que usa HTTP para transmitir mensajes.
HttpServerChannel	Http	Implementa un canal servidor (interfaz IChannelReceiver) que usa HTTP para transmitir mensajes.
TcpChannel	Tcp	Implementa un canal cliente y servidor (interfaces IChannelReceiver e IChannelSender) que usa TCP para transmitir mensajes.
TcpClientChannel	Tcp	Implementa un canal cliente (interfaz IChannelSender) que usa TCP para transmitir mensajes.
TcpServerChannel	Tcp	Implementa un canal servidor (interfaz IChannelReceiver) que usa TCP para transmitir mensajes.
IpcChannel	Ipc	Implementa un canal cliente y servidor (interfaces IChannelReceiver e IChannelSender) que usa IPC para transmitir mensajes.
IpcClientChannel	Ipc	Implementa un canal cliente (interfaz IChannelSender) que usa IPC para transmitir mensajes.
IpcServerChannel	Ipc	Implementa un canal servidor (interfaz IChannelReceiver) que usa IPC para transmitir mensajes.

y el servidor usan para intercambiar mensajes. Cuando un cliente hace una llamada a un objeto remoto, dicha llamada es serializada dentro de un mensaje. Éste es enviado por un canal cliente y recibido por un canal servidor; entonces se deserializa y procesa. Los valores de retorno son transmitidos por el canal servidor y recibidos por el canal cliente. El namespace **System.Runtime.Remoting.Channels** contiene las clases que soportan y manejan canales.

Los formateadores (*formatters*) son los objetos que gobiernan la forma en que se serializan y deserializan los objetos antes de ser transmitidos y luego de ser recibidos por los canales, respectivamente. La clase **BinaryFormatter** del namespace **System.Runtime.Serialization.Formatters.Binary** se usa para serializar y deserializar objetos en formato binario (propietario). La clase **SoapFormatter** del namespace **System.Runtime.Serialization.Formatters.Soap** se emplea para serializar y deserializar objetos en formato SOAP. Cada canal tiene asociado un formatter por default e, incluso, podemos configurarlos. También podemos crear nuestros propios canales y formateadores.

El modo de activación determina si el cliente o el servidor controlan el tiempo de vida (tiempo total durante el cual un objeto permanece activo en memoria) y el ámbito de los objetos remotos. Remoting soporta dos modos de activación para objetos de tipo MBR: la **Activación Cliente** (referida como Activated) hace que el cliente cree **Client-activated objects** (CAOs) en el servidor cuando llama a **new** o **Activator.CreateInstance**; y la **Activación Servidor** (referida como Well known) hace que el servidor cree **Server-activated objects** (SAOs) sólo cuando se necesita. El objeto proxy se crea al llamar a **Activator.GetObject**, pero el objeto servidor correspondiente no se crea hasta que el cliente invoca por primera vez un método sobre el proxy. A su vez, un SAO puede ser de tipo **SingleCall** o **Singleton**. Cuando se usan objetos **SingleCall**, el servidor crea una nueva instancia del objeto remoto por cada llamada a un método hecha por un cliente. Cuando la llamada se completa, marca el objeto para el garbage collector (recolector de basura). Cuando se usan objetos **Singleton**, una única instancia del objeto es creada en el servidor para manejar todas las llamadas de todos los clientes.

Tabla 20 | Formateadores por defecto

Canal	Formateador por defecto
HttpChannel	SoapFormatter
HttpClientChannel	SoapFormatter
HttpServerChannel	SoapFormatter
TcpChannel	BinaryFormatter
TcpClientChannel	BinaryFormatter
TcpServerChannel	BinaryFormatter
IpcChannel	BinaryFormatter
IpcClientChannel	BinaryFormatter
IpcServerChannel	BinaryFormatter

Integración de WCF y MSMQ

MSMQ permite la comunicación entre aplicaciones que pueden permanecer fuera de línea en forma asincrónica.

Las aplicaciones envían mensajes a colas (*queues*) y los leen en colas, de las que pueden eliminarlos o no. MSMQ está disponible en los sistemas operativos Windows NT, 2000, XP y 2003 como un componente adicional que se ejecuta como un servicio una vez instalado. Las colas de mensajes se mantienen en algún tipo de almacenamiento.

Pueden usarse *queues* de MSMQ para la comunicación entre aplicaciones *Windows Communication Foundation* (WCF). Esta capacidad hace que se pueda garantizar el envío de mensajes en forma independiente del tiempo de ejecución de los dominios de aplicación que los mandan y reciben. Existirá una diferencia de tiempo entre el momento en que una aplicación (*sender*) envía un mensaje y aquél en que otra (*receiver*) lo recibe. Este tiempo se denomina latencia. MSMQ implementa un protocolo nativo (*native*) para transferencias de *queue* a *queue*. También implementa un protocolo basado en SOAP, llamado SRMP (*Soap Reliable Messaging Protocol*), que se usa cuando en transferencias de *queue* a *queue* se emplea HTTP. SRMPS (*Soap Reliable Messaging Protocol Secure*) se utiliza cuando usamos HTTPS para las transferencias.

La clase **System.ServiceModel.NetMsmqBinding** es el binding provisto por el framework para que dos end-points de WCF se comuniquen usando MSMQ. En un archivo de configuración el elemento es **<netMsmqBinding>**. También podemos integrar aplicaciones *Message Queuing* (MSMQ) existentes con aplicaciones *Windows Communication Foundation* (WCF) usando *Message Queuing Integration Binding* para convertir mensajes MSMQ en mensajes WCF, y viceversa. Usando .NET, las aplicaciones MSMQ se construyen con las clases proporcionadas por el espacio de nombres **System.Messaging.**

El namespace **System.ServiceModel.MsmqIntegration** contiene clases que permiten la comunicación entre aplicaciones MSMQ y WCF. En particular, la clase **MsmqIntegrationBinding** permite comunicar un end-point WCF con una aplicación MSMQ (mapea mensajes MSMQ con mensajes WCF). En un archivo de configuración el elemento es **<msmqIntegrationBinding>**. El MSMQ Queue Manager es implementado como un servicio, y se ejecuta en background en las computadoras que envían y reciben mensajes.

PUEDEN USARSE QUEUES DE MSMQ PARA LA COMUNICACIÓN ENTRE APLICACIONES WCF. ESTA CAPACIDAD HACE QUE SE PUEDA GARANTIZAR EL ENVÍO DE MENSAJES EN FORMA INDEPENDIENTE DEL TIEMPO DE EJECUCIÓN DE LOS DOMINIOS DE APLICACIÓN QUE LOS MANDAN Y RECIBEN.

Seguridad en WCF

Es esencial garantizar la seguridad de los mensajes entre clientes y servicios. Veremos cómo efectuar esta tarea.

La seguridad en WCF está dividida en tres áreas funcionales:

- Seguridad de transferencia
- Control de acceso
- Auditoría

La seguridad de transferencia debe asegurarnos la integridad y la confidencialidad de los mensajes, como así también la autenticación de identidades. Los modos de implementar seguridad de transferencia en WCF se denominan **Transport**, **Message** y **TransportWithMessageCredential**.

El modo **Transport** implementa seguridad a nivel de protocolo de transporte (por ejemplo, usando SSL sobre HTTP). Si un mensaje necesita ir a través de múltiples puntos para alcanzar su destino final, cada punto intermedio debe hacer un forward del mensaje sobre una nueva conexión SSL (punto a punto).

La seguridad de transferencia debe asegurarnos la integridad de los mensajes, así como la autenticidad de las identidades.

El modo **Message** implementa seguridad a nivel de mensaje SOAP mediante el uso de especificaciones como WS-Security. Toda la información relativa a la seguridad se encapsula en el mensaje. La seguridad del mensaje se mantiene a través de cualquier cantidad de puntos intermedios entre el emisor origen y el destino final (final a final). Pueden ser firmadas digitalmente o encriptadas sólo partes de un mensaje, en vez del mensaje completo. Esto facilita que distintos intermediarios puedan ver partes del mensaje.

El modo **TransportWithMessageCredential** autentica el cliente usando seguridad a nivel de mensaje. Para autenticar el servicio, y asegurar la integridad y confidencialidad de los mensajes, se usa seguridad a nivel de

Tabla 21 | Aspectos por considerar sobre seguridad

Integridad de los mensajes	La alteración de un mensaje puede permitir obtener resultados diferentes de los deseados.
Confidencialidad de los mensajes	No debe permitirse la intercepción de mensajes que deje al descubierto información confidencial.
Autenticación del cliente	No permitir que un servicio realice tareas para quien no tiene permitido hacerlas.
Autenticación del servicio	No permitir que una entidad no autenticada actúe como un servicio e intercepte mensajes de los clientes para obtener información confidencial (ataque de phishing).
Repetición de mensajes	Debe detectarse si un mismo mensaje es enviado repetidas veces.

protocolo. Control de acceso se encarga de la autorización de los usuarios, y Auditoría se ocupa de grabar eventos de seguridad en el log de eventos de Windows.

Los bindings en WCF son objetos usados para especificar los detalles de comunicación (como protocolo de transporte o mecanismo de seguridad) requeridos para conectarnos a un end-point WCF. Podemos crear nuestros propios bindings (custom) o usar los provistos por el sistema (por ejemplo, **System.ServiceModel.WSHttpBinding**).

Para configurar el modo de seguridad utilizado, recurrimos a la propiedad **Mode** del objeto retornado por la propiedad **Security** del binding correspondiente. Veamos un ejemplo.

> Los bindings son objetos usados para especificar los detalles de comunicación requeridos para conectarnos a un end-point.

```
Dim b As New WSHttpBinding()
b.Security.Mode = SecurityMode.Transport
```

El objeto retornado por la propiedad **Security** del binding también posee las propiedades **Message** y **Transport**, que permiten determinar las configuraciones de seguridad a nivel de mensaje y transporte, respectivamente. Por ejemplo, la propiedad **ClientCredentialType**

Seguridad a Nivel de Protocolo

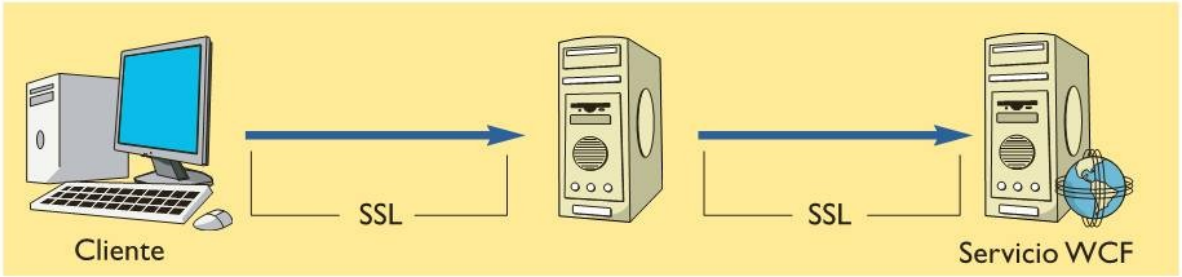


FIGURA 012 | Vemos cómo se aplica la protección a nivel de protocolo.

Tabla 22 | Enumeración del objeto SecurityMode

Miembro	Descripción
None	Seguridad deshabilitada.
Transport	Seguridad a nivel de protocolo de transporte.
Message	Seguridad a nivel de mensaje SOAP.
TransportWithMessageCredential	Seguridad a nivel de protocolo de transporte para integridad, confidencialidad y autenticación. Seguridad a nivel de mensaje para autenticación del cliente.

de Transport para el binding WSHttpBinding (por defecto, None) puede configurarse con alguno de los valores de la enumeración **HttpClientCredentialType** (Basic, Certificate, Digest, None, Ntlm o Windows).

En código, en Visual Basic:

```
Dim b As New WSHttpBinding()
b.Security.Mode = SecurityMode.Transport
b.Security.Transport.ClientCredentialType =
HttpClientCredentialType.Windows
```

Usando archivo de configuración:

```
<wsHttpBinding>
<binding name="TransportSecurity">
    <security mode="Transport" />
        <transport clientCredentialType =
        "Windows" />
    </security>
</binding>
</wsHttpBinding >
```

Por defecto, todos los bindings provistos por el sistema tienen habilitado algún modo de seguridad, excepto **System.ServiceModel.BasicHttpBinding** (por defecto, **Security.Mode** tiene como valor el **BasicHttpSecurityMode.None**).

Seguridad a Nivel de Mensaje

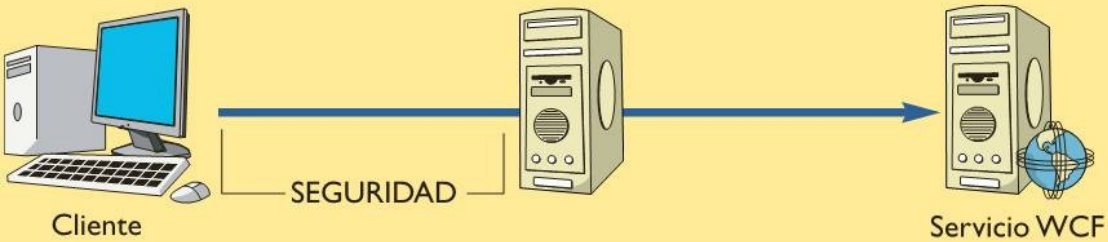


FIGURA 013 | En este caso, la protección se aplica a nivel de mensaje.

Tabla 23 | Modos de seguridad por defecto

Tipo de enlace	Modo de seguridad por defecto
BasicHttpBinding	None
WSHttpBinding	Message
WSDualHttpBinding	Message
WSFederationHttpBinding	Message
NetTcpBinding	Transport
NetNamedPipeBinding	Transport
NetMsmqBinding	Transport
NetPeerTcpBinding	Transport
MsmqIntegrationBinding	Transport

Windows Workflow Foundation

Diseño e implementación 15

Contenidos

En este capítulo aprenderemos sobre esta herramienta llamada Windows Workflow Foundation, que está integrada en el nuevo Framework de Microsoft. Veremos cómo crear un workflow, implementarlo y las ventajas de su uso.

Temas tratados

- Conceptos generales
- Arquitectura
- Servicios Runtime
- Creación de workflows
- Compensación de workflows

Windows Workflow Foundation

Conoceremos este nuevo concepto que cada vez es más popular dentro del mundo del desarrollo.

Conceptos generales

Muchos de nuestros trabajos consisten en resolver problemas de negocios de diversas actividades. Su complejidad dependerá de la naturaleza de la actividad.

- Definición
- Flujos de trabajo en la vida cotidiana
- Herramientas que utilizaremos
- Diferentes flujos de trabajo

Arquitectura

Aprenderemos en qué escenarios puede usarse Windows Workflow Foundation y para qué plataformas se encuentra disponible. Repasaremos su estructura de clases y sus componentes.

- Escenarios de usos
- Librería de clases
- Componentes
- Runtime Engine y Runtime Services

Servicios Runtime

Aprenderemos a utilizar los servicios de tiempo de ejecución que nos brinda Windows Workflow Foundation, como así también, para qué propósitos emplearlos.

- Servicios de tareas programadas
- Servicios de persistencia
- Servicios de monitoreos
- Servicios transaccionales

Creación de workflows

Aprenderemos a utilizar los diferentes diseñadores que vienen con esta herramienta, y veremos cómo crear flujos de trabajo y extenderlos a través de actividades personalizadas.

- Reglas
- Condiciones
- Políticas
- Actividades personalizadas

Compensación de workflows

Conoceremos, además, cómo interconectar un flujo de trabajo con otro utilizando diferentes tecnologías, y aprenderemos a utilizar transacciones en ellos.

- Transacciones
- Trabajando con workflows
- Métodos y eventos
- Workflows y Servicios Web

Windows Workflow Foundation

Veremos de qué se trata exactamente y cómo implementarlo para llevar adelante nuestros desarrollos.

Workflow es la automatización de un proceso de negocios, en forma completa o parcial, durante la cual los documentos, la información o las tareas se pasan de un participante a otro para ejecutar una acción, según un conjunto de reglas de procedimiento. La definición precedente proviene de la organización *Workflow Management Coalition* (WfWC), que desarrolla estándares para proveer de interoperabilidad entre los sistemas de workflow y las aplicaciones, como así también, entre diferentes sistemas de workflow.

También establece lo que es un *Workflow Managemet System* (WFMS), como un sistema que define, crea y maneja la ejecución de workflows a través del uso de software, corriendo en uno o más motores de workflow, que pueden interpretar la definición del proceso, interactuar con los participantes del workflow y, si se requiere, invocar el uso de herramientas de TI (tecnologías de la información) y aplicaciones.

Mucho de nuestro trabajo como desarrolladores consiste en resolver problemas de negocios de diversas actividades (bancarias, financieras, manufactureras, etc.). La complejidad de dichos procesos de negocios dependerá de la naturaleza de la actividad. Mas allá de lo complejo que sea el problema, para

Workflow Elemental

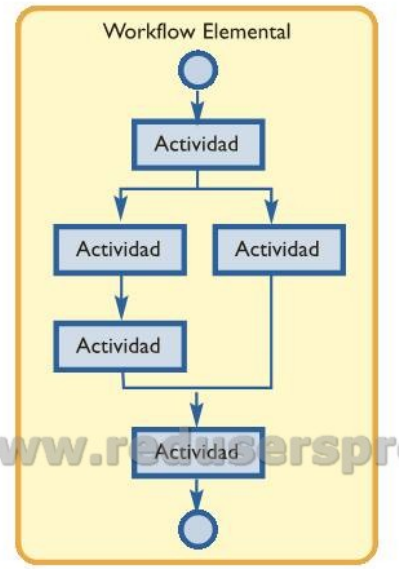


FIGURA 001 | Vemos cómo sería el caso de un workflow elemental.

resolverlo, tenemos que dividirlo en tareas pequeñas, comprensibles y manejables, cuya ejecución deberá producirse en un orden determinado y en conformidad con ciertas reglas. La implementación de la solución puede hacerse con un lenguaje de propósito general (como C# o Visual Basic). Nuestras líneas de código especificarán qué hacer, en qué secuencia, y tomarán decisiones basadas en valores de variables, eventos y el estado de la aplicación. En forma consciente o no, hemos descrito un workflow. La mayoría de nuestros proyectos para implementar un proceso de negocios cuenta con casos de uso y/o diagramas UML; es decir, con modelos que permiten visualizar el proceso. También solemos usar diagramas informales en el sentido de que no tienen una semántica formalmente definida. Pero lo que necesitamos son herramientas gráficas, de modo que, mientras modelamos el workflow, también se esté construyendo la aplicación.

Consideremos el siguiente ejemplo:

- Una persona concurre a un banco para solicitar un crédito.
- Un oficial de negocios del banco chequea su historial crediticio.
- Basado en dicho historial, el oficial ofrece diferentes líneas de crédito posibles o informa a la persona que no califica para obtener un crédito.
- Si califica, la persona elige una de las opciones.
- El oficial de negocios inicia el expediente y solicita a la persona determinada documentación.
- El oficial de negocios preaprueba el expediente y lo envía al oficial de riesgo crediticio.
- El oficial de riesgo crediticio aprueba o rechaza el crédito basado en información adicional sobre el solicitante y el sistema financiero.
- Si el crédito es aprobado, el dinero se transfiere a la cuenta que la persona indique.
- Si el crédito es rechazado, la persona es informada de tal suceso.

Recepción de Muestras en Laboratorio

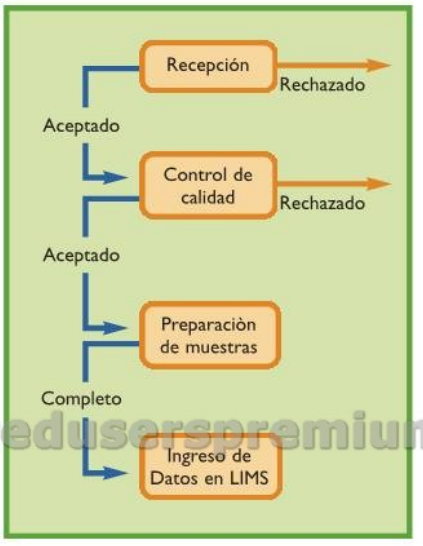


FIGURA 002 | Ejemplo de un workflow de personas.

Como vemos, el control y una parte de la ejecución del proceso son realizados por personas. El oficial de negocios obtendrá el historial crediticio de la persona con el banco basándose en un sistema de información interno. Pero requerirá de datos o eventos que llegan desde el exterior para conocer el historial crediticio con otras entidades; esto involucra la comunicación asincrónica con otros sistemas. En ocasiones, debemos esperar a que se produzcan datos o eventos para poder continuar realizando un trabajo. Esto ocurre cuando pedimos a la persona documentación que, posiblemente, no tenga consigo. Este hecho hace que nuestro flujo de trabajo quede inactivo mientras se esperan esos datos, lo cual puede demandar horas, días o, incluso, meses. Por lo tanto, nuestro flujo de trabajo deberá ser persistido hasta el momento en que se produzcan los datos o eventos necesarios para continuar. Una vez persistida la instancia de flujo de trabajo, deberemos removerla de memoria. Al recibir los datos, tenemos que restaurarla y continuar la ejecución.

¿Qué pasa si el proceso de negocios es modificado antes de que las solicitudes pendientes terminen de ser procesadas? Podemos resolver todo el proceso usando un lenguaje de propósito general (como C# o Visual Basic), pero tendremos que escribir mucho código de soporte para la ejecución del workflow. Quisiéramos especificar el workflow de forma declarativa –si es posible, gráficamente– y contar con un motor de workflow que interprete dicha definición e implemente los pasos necesarios para completar la tarea.
A su vez, a medida que se diseñan nuevos procesos de negocios, éstos deben ser implementados. Así, el desarrollo de software genérico para el manejo de procesos de negocios, llamados *Workflow Management Systems* (WFMS), cobra importancia. La idea de contar con ese tipo de herramientas genéricas se remonta a los años '70 con precursores como Skip Ellis y su grupo de trabajo en Xerox.

Los componentes de workflow son usados para controlar la secuencia de distintas funciones.

Los *Workflow Management Systems* (WFMS) pueden ser autónomos o embebidos. Los primeros son aplicaciones que proveen de la funcionalidad de workflow sin necesidad de tener otra aplicación, excepción hecha de un sistema de manejo de bases de datos (por ejemplo, SQL Server) y un sistema de cola de mensajes (como MSMQ). Los segundos son funcionales sólo si se los usa dentro del contexto de otra aplicación; por ejemplo, un sistema ERP (*Enterprise Resource Planning*). En este caso, los componentes de workflow son usados para controlar la secuencia de distintas funciones de la aplicación, así como para manejar colas de mensajes, notificaciones, eventos y excepciones. *Microsoft Windows Workflow Foundation* (WF) fue creado para cubrir este tipo de requerimientos como un componente núcleo del .NET Framework 3.0. Es declarativo, visual y flexible. El modelo de programación que propone separa lo que tenemos que hacer (nuestra lógica de negocios) del momento en que debe hacerse (nuestro flujo de trabajo). Esto permite modificar el modelo del workflow sin afectar la lógica de negocios. Al desarrollar, definiremos el modelo de nuestro workflow como un mapa de actividades, y lo compilaremos como un assembly .NET que será ejecutado por el runtime de workflow y el *Common Language Runtime* (CLR).
Existen diferentes tipos de workflows: de personas y de sistemas. Los **workflows de personas** implican coordinar procesos de negocio que involucran la interacción de individuos. Diseñar workflows de este tipo eficaces requiere comprender en detalle los requisitos técnicos de los pasos individuales del proceso, así como de la forma en que los usuarios finales realizan esos

pasos. De dicha comprensión dependerá el desarrollo de una solución acertada. Un ejemplo típico es el de un laboratorio de análisis químicos. En este caso, las personas (los analistas) interactúan entre sí, con software y con máquinas especializadas. Entre otros detalles, la preparación, la registración y el almacenamiento de las muestras ingresadas en el laboratorio dependerán de los tipos de análisis que se deban realizar, de las máquinas que se utilicen y de las técnicas de determinación empleadas. Para soportar la comunicación entre personas y sistemas, los workflows de personas tienen que brindar funciones básicas de asignación de tareas, manejo de identidades, notificaciones, seguimiento e interoperabilidad con sistemas de gerenciamiento del negocio-proceso (*business-process management* o BPM).
La pregunta que surge es si los Workflow Management Systems y los BPM son lo mismo. En el sentido en que ambos se centran en el manejo del proceso, sí. Tradicionalmente, se consideran como BPM los productos optimizados para los procesos que son automáticos, que no implican la interacción humana. Es decir, **workflows de sistemas**.
También suele encontrarse que estos productos se denominan de Integración de Aplicaciones Empresariales (*Enterprise Application Integration* o EAI), siendo un ejemplo Microsoft BizTalk. Los productos de tipo Workflow Management Systems se consideran optimizados para controlar la incertidumbre y el retraso asociados a la interacción de las personas. En el mercado también se habla de BPM como el tipo de sistemas que cubre ambos workflows –de personas y de sistemas–, y se los denomina BPMS (BPM Suite).
Microsoft usa la palabra workflow en su *Windows Workflow Foundation* (WF) para describir una tecnología central que permite construir workflows. Un workflow de sistemas o un workflow de personas describe formas diferentes de usar la misma tecnología.

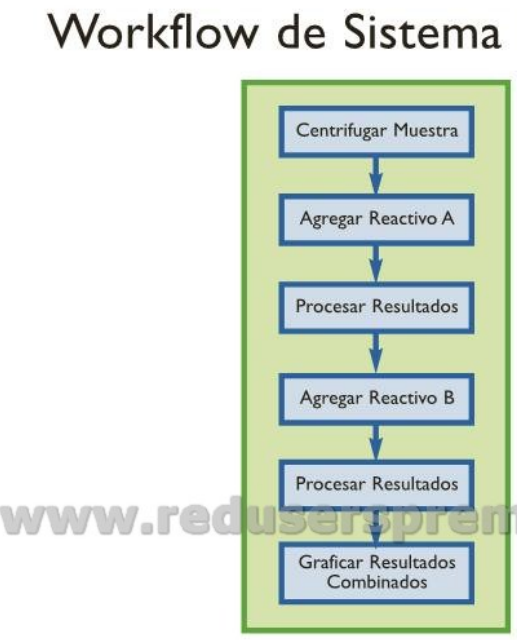


FIGURA 003 | Ejemplo de un workflow de sistema.

Arquitectura y componentes de WWF

WWF otorga a los desarrolladores un motor de workflow genérico para construir flujos de trabajo.

Windows Workflow Foundation puede ser usado en distintos escenarios para coordinar la interacción entre aplicaciones, personas o ambos. Ejemplos de ello son el manejo del ciclo de vida de documentos dentro del contexto de la certificación de una norma en una empresa (como la ISO); el desarrollo de un sistema BPM; o el workflow de una aplicación de compras e inventario que requiera aprobación de proveedores, presupuestos, requerimientos de compra y órdenes de compra. WF no es una aplicación de workflow para usuarios finales.

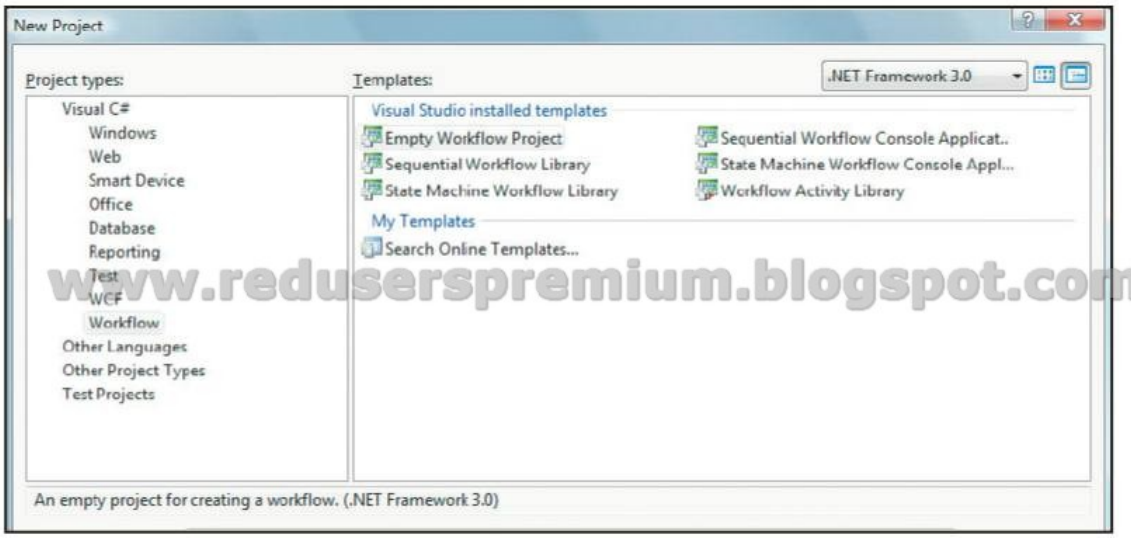
Windows Workflow Foundation soporta los modelos de workflow secuencial (*sequential*) y máquina de estado (*state machine*). El diseñador de Windows Workflow Foundation disponible para Visual Studio 2005/2008 (en 2005, a través de la instalación de las extensiones para WWF) provee a los desarrolladores de un entorno de trabajo altamente productivo. Podemos construir workflows gráficamente con el diseñador, usando un dialecto XML específico, código en lenguajes .NET o una combinación de todo. Finalmente, se compila en un assembly .NET estándar. Una vez instaladas las extensiones, dispondremos del tipo de proyecto **Workflow** con varias plantillas, como se muestra en la Figura 4.

Las plantillas Sequential Workflow Console Application y State Machine Workflow Console Application brindan el modelo de workflow para dos tipos de flujo que pueden crearse: secuencial (*sequential*) y máquina de estado (*state machine*).

Veamos a continuación algunos elementos y características de WWF:

- Librería de clases: el .NET Framework 3.0 nos provee de las clases base e interfaces necesarias para construir aplicaciones de workflow,

FIGURA 004 | Cuadro de diálogo Nuevo Proyecto, en Visual Studio 2008.



Las plantillas Sequential Workflow Console Application y State Machine Workflow Console Application brindan el modelo para dos tipos de flujo: secuencial y máquina de estado.

organizadas en varios namespaces System.Workflow, los cuales se presentan en la Tabla 1.

- Librería de actividades básicas: Windows Workflow Foundation incluye un conjunto de actividades para el diseño de workflows denominado *Base Activity Library* (BAL). El conjunto de clases que las definen está incluido en el namespace **System.Workflow.Activities**, y brindan funcionalidades tales como control de flujo, condiciones, manejo de eventos, manejo de estados, y comunicación con aplicaciones y

Tabla 1 | System.Workflow namespaces

Namespace	Descripción
System.Workflow.Activities	Contiene clases que definen las actividades incluidas en WF para construir workflows, por ejemplo, la actividad Code.
System.Workflow.Activities.Configuration	Provee clases que representan secciones del archivo de configuración.
System.Workflow.Activities.Rules	Conjunto de clases que definen las condiciones y acciones que conforman una regla.
System.Workflow.Activities.Rules.Design	Conjunto de clases que manejan los cuadros de diálogo Rule Set Editor y Rule Condition Editor de Visual Studio.
System.Workflow.ComponentModel	Provee las clases base, interfaces y el núcleo de construcciones para el modelado, que son usadas para crear actividades y workflows. Contiene la clase base de todas las actividades: Activity.
System.Workflow.ComponentModel.Compiler	Provee infraestructura para la validación y compilación de actividades y workflows.
System.Workflow.ComponentModel.Design	Contiene clases que permiten a los desarrolladores extender el comportamiento de tiempo de diseño para workflows y actividades.
System.Workflow.ComponentModel.Serialization	Provee la infraestructura para manejar la serialización de actividades y workflows hacia y desde Extensible Application Markup Language (XAML) y CodeDOM.
System.Workflow.Runtime	Contiene clases e interfaces para controlar el motor de runtime y la ejecución de una instancia de workflow.
System.Workflow.Runtime.Configuration	Contiene clases que se usan para configurar el motor de runtime.
System.Workflow.Runtime.DebugEngine	Contiene clases e interfaces para usar en el debug de instancias de workflow.
System.Workflow.Runtime.Hosting	Contiene clases relacionadas a los servicios provistos por la aplicación host al motor de runtime.
System.Workflow.Runtime.Tracking	Contiene clases e interfaces relacionadas con el servicio de tracking.

servicios. Están disponibles en el Toolbox del diseñador en Visual Studio.

- Motor de tiempo de ejecución (Runtime Engine): el motor de runtime se encarga de la ejecución de los workflows y del manejo de su estado a lo largo de todo su ciclo de vida. Provee un entorno de ejecución para los workflows. No es una aplicación autocontenida, y está representado por la clase **WorkflowRuntime** del namespace **System.Workflow.Runtime**. Debemos proporcionar una aplicación host que será responsable de la creación de una instancia de la clase **WorkflowRuntime**, y obtendremos la ejecución *in-process* del motor dentro del host. Los tipos de aplicaciones que pueden actuar como host son:

- Aplicaciones Windows de consola
- Servicios Windows (por ejemplo, Windows SharePoint Services 3.0)
- Aplicaciones Windows Forms (WinForms)
- Aplicaciones WPF
- Aplicaciones Web ASP.NET
- Servicios Web ASP.NET

- Servicios de tiempo de ejecución (Runtime Services): el motor de runtime extiende su funcionalidad mediante el uso de servicios externos que cumplen propósitos determinados. Estos servicios son instancias de clases que se crean y registran con el motor de runtime desde la aplicación host. Windows Workflow Foundation otorga cuatro servicios, denominados core, que son:

- Scheduling
- CommitWorkBatch
- Persistence
- Tracking

Los servicios denominados core sólo pueden agregarse al motor de ejecución (método **AddService** de la clase WorkflowRuntime) antes de que éste arranque (método **StartRuntime** de la clase WorkflowRuntime).

Workflow de sistema

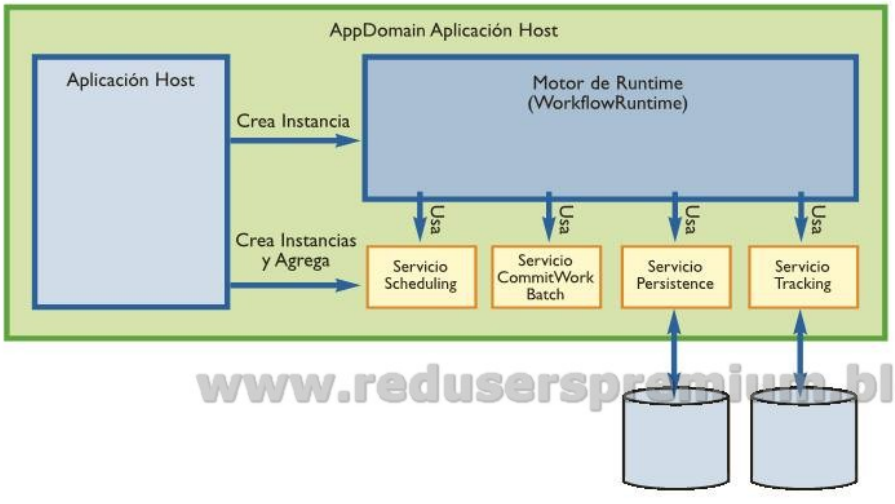


FIGURA 005 | Vemos cómo trabaja un workflow de sistema.

Definición de actividades

Una actividad es el bloque básico de construcción de un workflow en WF. Todas las actividades derivan de la clase base **System.Workflow.ComponentModel.Activity**. WF nos provee de un conjunto de actividades para el diseño de workflows denominado *Base Activity Library* (BAL). El conjunto de clases que las definen está incluido en el namespace **System.Workflow.Activities**. El toolbox del Workflow Designer en Visual Studio 2005 contiene iconos para las actividades en la BAL. Las actividades que aparecen en el toolbox varían según el tipo de workflow que estemos modelando: secuencial o de máquina de estado. Las actividades se clasifican en simples (básicas) y compuestas (*composite*). Simples son aquellas cuya lógica y ejecución se encapsulan dentro del código de la actividad, como ejecutar una sentencia SQL. Las compuestas contienen otras actividades (denominadas hijas) y dependen de la ejecución de todas ellas para alcanzar su propósito funcional.

Podemos crear nuestras propias actividades, tanto básicas como compuestas. Para hacerlo, creamos un proyecto de tipo Workflow en Visual Studio usando el template **Workflow Activity Library**. Obtenemos una actividad por default (representada por los archivos Activity1.vb y Activity1.Designer.vb si usamos Visual Basic), cuya clase base es **System.Workflow.Activities.SequenceActivity**. Este tipo de actividad compuesta reusa el control de ejecución de la clase base SequenceActivity (la cual deriva de **CompositeActivity**). Si queremos tener mayor control sobre la ejecución de las actividades hijas, nos conviene derivar nuestra actividad compuesta de la clase base **CompositeActivity** y sobrescribir su método **Execute**. Este método recibe como parámetro un objeto de tipo **System.Workflow.ComponentModel.ActivityExecutionContext** cuando es invocado por el motor de runtime para ejecutar una actividad. Un **ActivityExecutionContext** (AEC) representa el entorno de ejecución de una actividad. Se crea implícitamente un AEC cuando la aplicación host invoca el método **WorkflowInstance.Start**. Para cambiar la clase base de la cual derivaremos nuestra actividad, usamos la Ventana de Propiedades. En ella seleccionamos la propiedad **Base Class** y hacemos clic en el botón

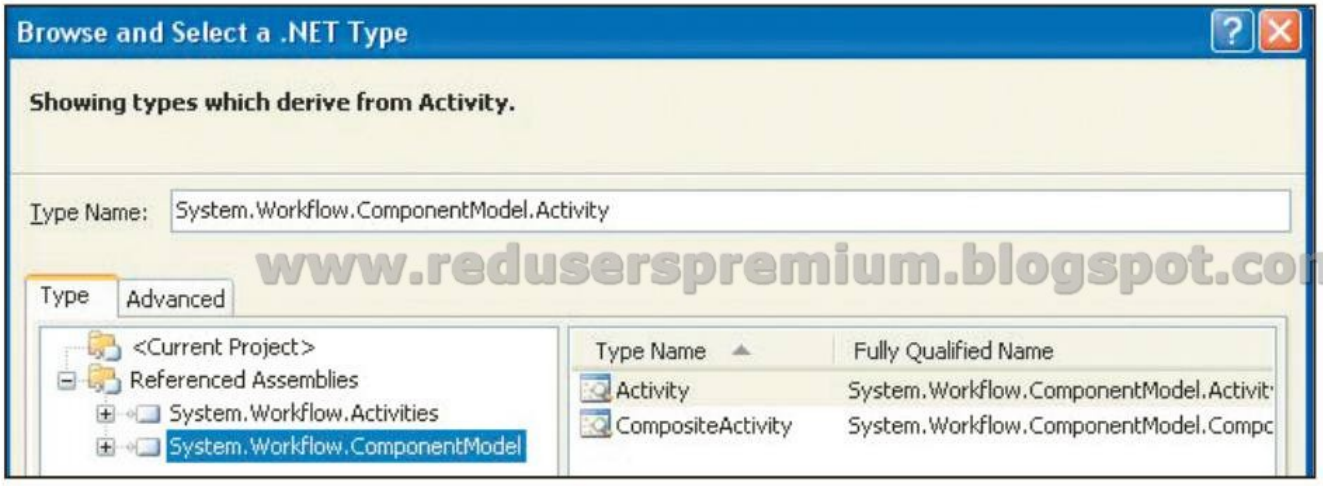


FIGURA 006 | Cuadro de diálogo para buscar y seleccionar un tipo en .NET.

"..." asociado. Aparece el cuadro de diálogo **Browse and Select a .NET Type**, que nos permite seleccionar una clase base.

El código por defecto en el archivo Activity1.vb es similar al siguiente

```
Public Class Activity1
    Inherits System.Workflow.Activities.
    SequenceActivity
End Class
```

Éste se modificará de la siguiente manera si cambiamos a una clase base **CompositeActivity**:

```
Public Class Activity1
    Inherits System.Workflow.ComponentModel.
    CompositeActivity
End Class
```

Propiedades de actividades

La clase Activity deriva de **System.Workflow.ComponentModel.DependencyObject**, de modo que pueden definirse propiedades estándar del CLR o propiedades de dependencia (clase **System.Workflow.ComponentModel.DependencyProperty**) al crear una actividad. Hay dos tipos de propiedades de dependencia en las actividades de WF: de metadatos y de instancia.

La clase **DependencyObject** participa en el sistema de dependencia de propiedades (*Dependency Property System*), cuya función primaria es computar los valores de las propiedades y otorgar un sistema de notificación sobre el cambio de los valores. En este contexto, la clase **DependencyProperty** permite la registración de las propiedades de dependencia dentro del sistema de dependencia de propiedades a través del método **Register**. El valor de una propiedad de dependencia se almacena en un repositorio central. Las propiedades de dependencia nos ofrecen una funcionalidad importante en WF denominada **Activity binding**. Como su nombre lo indica, nos permite hacer el binding de una propiedad en una actividad, con una propiedad en otra actividad, o con una propiedad en el workflow. Esto hace que el valor de la propiedad se propague en forma automática. De lo contrario, deberíamos programar todas las asignaciones de valores necesarias.

Para usar propiedades de dependencia en nuestras actividades seguimos un patrón. Agregamos una instancia de DependencyProperty que sea Public Shared (PublicStatic en C#) para manejar la propiedad en todas las instancias de la actividad. Luego, agregamos una propiedad estándar con Get y Set. En el Get usamos el método GetValue definido en la clase DependencyObject, para obtener el valor almacenado de la propiedad. En el Set usamos el método SetValue definido en la clase DependencyObject para almacenar el valor de la propiedad. Un code snippet nos ayuda a crear fácilmente propiedades de dependencia.

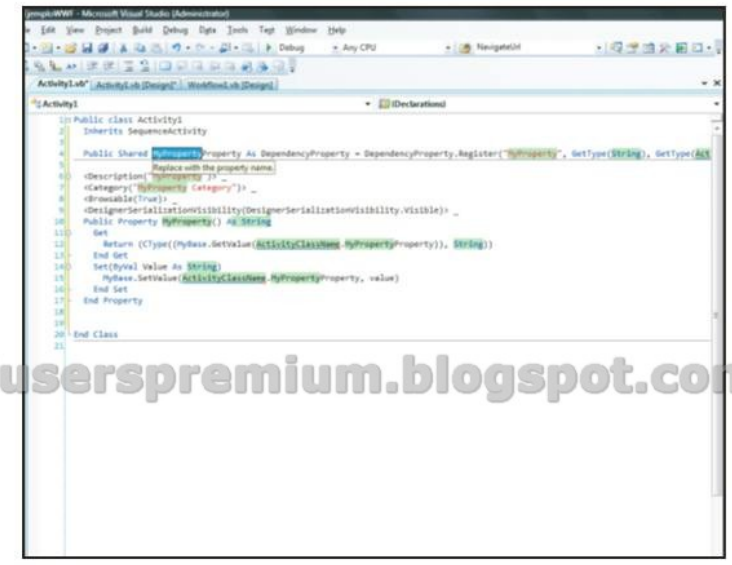


FIGURA 007 | Code snippet en acción.

Introducción a los diseñadores de workflows

Para diseñar workflows en Visual Studio 2005 debemos instalar las extensiones para Windows Workflow Foundation (disponibles para su descarga en el sitio de Microsoft); en Visual Studio 2008, ya vienen instaladas por defecto. Tendremos disponible el tipo de proyecto Workflow con los templates que se presentan en la Tabla 3.

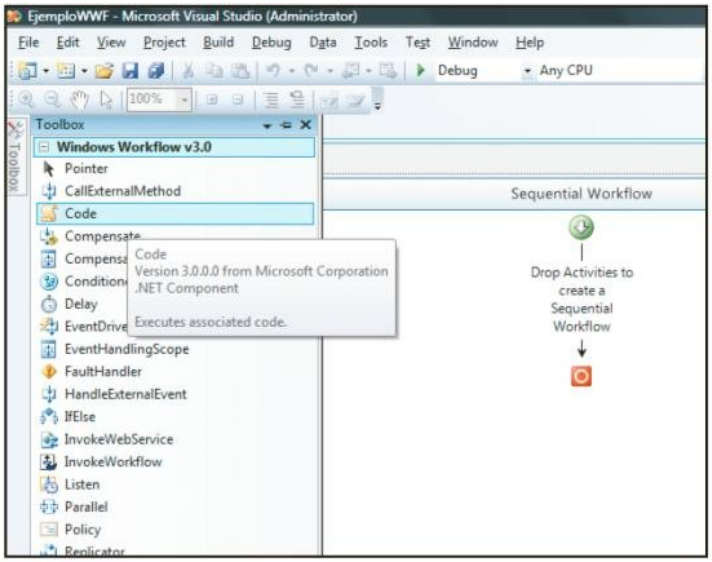


FIGURA 008 | Agregar una actividad Code.

Modos de autoría

Windows Workflow Foundation soporta los siguientes modos de autoría para la implementación de workflows:

- Code-only
- Code-separation
- No-code

El modo **code-only** es el usado por default en el entorno de desarrollo de Visual Studio. En este modelo, la definición del workflow se hace por completo en un lenguaje como C# o Visual Basic .NET usando el modelo de objetos de WF. Por ejemplo, vamos a crear un proyecto de tipo Workflow con el template **Sequential Workflow Console Application** usando Visual Basic. En este caso, se generan dos archivos; **Workflow1.vb** contiene por default el siguiente código:

VB:

```
Public class Workflow1
    Inherits SequentialWorkflowActivity
End Class
```

En este archivo se agrega código para manejo de eventos y lógica de negocios.

Tabla 2 | Templates para el proyecto Workflow

Tipo	Descripción
Sequential Workflow Console Application	Crea un proyecto para construir workflows que contiene un workflow secuencial por default y una aplicación de consola como host para test.
Sequential Workflow Library	Crea un proyecto de tipo librería de clases para construir una librería de workflows secuenciales sin desarrollar una aplicación host.
Workflow Activity Library	Crea un proyecto de tipo librería de clases para construir una librería de actividades que puedan reusarse en la construcción de workflows.
State Machine Console Application	Crea un proyecto para construir un workflow de tipo maquina de estado (state machine) y una aplicación de consola como host.
State Machine Workflow Library	Crea un proyecto de tipo librería de clases para construir una librería de workflows de tipo maquina de estado (state machine).
Empty Workflow	Crea un proyecto vacío que puede incluir workflows y actividades.

También vemos en el proyecto el archivo **Workflow1.Designer.vb**, que contiene el modelo de workflow mantenido por el diseñador. Luego, agregamos desde la toolbox una actividad Code en nuestro workflow, como se ve en la Figura 9.

El diseñador insertara código en **Workflow1.Designer.vb** para reflejar este cambio en el diseño. Si hacemos doble clic sobre la actividad, se crea el código en **Workflow1.vb**:

```
Private Sub codeActivity1_ExecuteCode(ByVal
sender As System.Object, ByVal e As
System.EventArgs)
End Sub
```

Y este otro en Workflow1.Designer.vb:

```
AddHandler Me.codeActivity1.ExecuteCode,
AddressOf Me.codeActivity1_ExecuteCode
```

Esto demuestra el uso del evento **Execute Code**, que ocurre cuando la actividad Code se inicia. A continuación, agregamos el siguiente código en el método manejador del evento y ejecutamos la aplicación.

VB:

```
Console.ForegroundColor = ConsoleColor.Green
Console.WriteLine("Hola Mundo!")
Console.ReadLine()
Console.ResetColor()
```

Los archivos mencionados definen la clase Workflow1 (observar el uso de la palabra clave **partial** en el archivo Workflow1.Designer.vb). Al construir (*build*) la solución, el compilador genera el tipo Workflow1 combinando el contenido de ambos archivos.
Cabe notar que la clase Workflow1 deriva de la clase SequentialWorkflowActivity. Esta última deriva de la clase SequenceActivity, que, a su vez, deriva de CompositeActivity, que deriva de Activity. Así es que los workflows en sí mismos están implementados como actividades.

Otro modo de autoría en WF se denomina **code-separation** o **code-beside**, y resultará familiar para quienes desarrollan con ASP.NET. En este caso, el modelado del workflow será mantenido por el diseñador en un archivo XML (con extensión .xoml) que declara el workflow usando XAML (*Extensible Application Markup Language*). XAML (markup) nos permite especificar instancias de objetos como elementos XML, y las propiedades de dichos objetos, como atributos XML. Un segundo archivo, referido como code-beside, se crea para contener código de manejo de eventos y lógica de negocios. Para usar este modo, creamos un workflow empleando el template **Sequential Workflow (with code separation)** del cuadro de diálogo **Add New Item**. Se generan dos archivos.
Workflow2.xoml.vb contiene el siguiente código (el mismo que **Workflow1.vb**):

```
Public Class Workflow2
    Inherits SequentialWorkflowActivity
End Class
```

Y **Workflow2.xoml** declara el modelo como:

```
<SequentialWorkflowActivity
x:Class="SequentialWorkflowConsoleApplication.
 Workflow2"
Name="Workflow2"
```

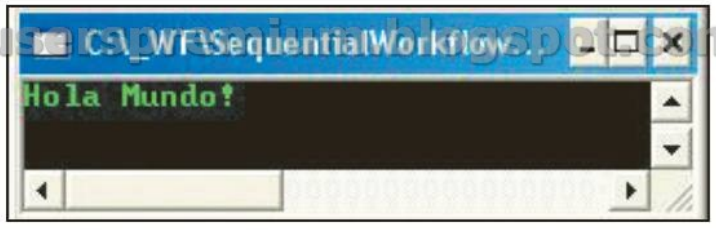


FIGURA 009 | Resultado de la ejecución.

```
xmlns="http://schemas.microsoft.com/winfx/2006/
xaml/workflow" xmlns:x="http://schemas.
microsoft.com/winfx/2006/xaml">
</SequentialWorkflowActivity>
```

El atributo x:Class apunta a la clase definida en **Workflow2.xoml.vb**. Si usando el diseñador, agregamos una actividad Code y hacemos doble clic sobre ella, en **Workflow2.xoml** se agrega este elemento:

```
<CodeActivity x:Name="codeActivity1"
ExecuteCode="codeActivity1_ExecuteCode" />
```

En este caso, al construir la solución, el compilador interpreta el archivo .xoml y genera en forma temporal una clase parcial en Visual Basic. Ésta es combinada con el archivo code-beside .xoml.vb para generar el tipo Workflow2.

La única línea de código que tenemos que cambiar de la aplicación host es la siguiente:

```
workflowInstance = workflowRuntime.CreateWork
flow(GetType(Workflow2))
```

Ambos modos de autoría nos ofrecen facilidad de uso en un entorno eficiente y con el cual los desarrolladores .NET ya están familiarizados. El modo code-separation es más flexible.

El tercer modo de autoría posible se denomina **no-code** (o **markup-only**). En este caso, no hay ningún archivo code-beside, y toda la definición del workflow se realiza utilizando XAML. No hay ningún template en Visual Studio que nos permita trabajar en este modo. Lo que sí podemos utilizar para crear nuestros archivos XAML es el editor XML incluido en Visual Studio.

FIGURA 010 | Archivos de la Solución Ejemplo.

Vamos a agregar un nuevo ítem de tipo archivo XML en nuestra solución, nombrándolo Workflow3.xoml. Si copiamos en este archivo el contenido de **Workflow2.xoml** (cambiando el nombre del workflow), podremos ver en el diseñador el mismo modelo. Como queremos implementar el workflow completamente en XAML, pondremos el código Visual Basic en un bloque CDATA dentro de un elemento x:Code, como se muestra a continuación:

XAML:

```
<SequentialWorkflowActivity
  x:Class="SequentialWorkflowConsole
  Application.Workflow3"
  x:Name="Workflow3"
  xmlns:x="http://schemas.microsoft.com/
  winfx/2006/xaml"
  xmlns="http://schemas.microsoft.com/
  winfx/2006/xaml/workflow">
  <CodeActivity x:Name="codeActivity1"
  ExecuteCode="codeActivity1_ExecuteCode" />
  <x:Code>
    <![CDATA[
    Private Sub codeActivity1_ExecuteCode
    (ByVal sender As System.Object, ByVal e
    As System.EventArgs)
        Console.ForegroundColor =
        ConsoleColor.Green
        Console.WriteLine("Hola Mundo 3!")
        Console.ReadLine()
        Console.ResetColor()
        End Sub
          ]]>
  </x:Code>
</SequentialWorkflowActivity>
```

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

22

Windows Communication Foundation

Service, Data y Message Contracts / Binding / MSMQ: Conceptos e integración / Web Service Enhancements / Remoting

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ISBN 978-987-1347-43-8
00022
9 789871 347438